Implementation of an approach to mitigate Yo-Yo attack in cloud auto-scaling mechanism

Meraj Mostamand Kashi

Thesis submitted for the degree of
Master in Applied Computer and Information Technology - ACIT
(Cloud-based Services and Operations)
30 credits

Department of Computer Science
Faculty of Technology, Art and Design

Oslo Metropolitan University — OsloMet

Spring 2022

Implementation of an approach to mitigate Yo-Yo attack in cloud auto-scaling mechanism

Meraj Mostamand Kashi

Implementation of an approach to mitigate Yo-Yo attack in cloud auto-scaling mechanism

http://www.oslomet.no/

Printed: Oslo Metropolitan University — OsloMet

Abstract

In recent years, global business has witnessed significant cloud adoption, which provides considerable value over traditional datacenters—achieving greater scalability, cost efficiency, and improved performance. Cloud auto-scaling is a cloud service feature to react to the variation in the live traffic load by spinning up or down instances on the fly. This new feature may also introduce new security threats. For example, DDoS attacks utilize multiple distributed attack resources to exploit resources such as cloud services. Auto-scaling mechanism transforms the DDoS attacks into Economic Denial of Sustainability attack (EDoS) or an emerging new type of attack called Yo-Yo attack. Yo-Yo attack is a newly disclosed attack, according to which attackers send a burst of traffic periodically to oscillate the auto-scaling system between scale-out and scale-in status.

In this thesis, we present a solution to detect a Yo-Yo attack and mitigate it in the cloud auto-scaling mechanism. The study shows to which extent the Yo-Yo attack differs from traditional DoS/DDoS attacks in cloud auto-scaling. An approach — called Trust-based Adversarial Scanner Delaying (TASD), which is introduced by [70] — is implemented and tested under real cloud settings. The TASD system is deployed on Amazon Web Services (AWS). In TASD, the detection module uses a trust value algorithm to assign a Quality of Service (QoS) value to each user, and the mitigation module controls the flow of the traffic base on the trust value number of each user.

The experimental results show that the Yo-Yo attack causes significant performance degradation in addition to economic damage, while the attack is more difficult to detect and requires fewer resources from the attacker compared with traditional DDoS. Moreover, auto-scaling policy configuration is a key to minimizing the effect of Yo-Yo attacks. The experiment evaluations show that the TASD system can detect and mitigate Yo-Yo attacks in a real cloud application.

Acknowledgments

First and foremost, I am extremely grateful to my supervisors, Hårek Haugerud and Anis Yazidi for their invaluable advice, continuous support, and patience during my master thesis study. Their immense knowledge and plentiful experience have encouraged me all the time in my academic research and hopefully, our paths will cross again.

I would also like to thank my wife, Shahrzad, who has stood by me through this research. Without her mental and motivational support, I may never have completed this thesis.

Finally, I must thank Oslo Metropolitan University for giving me the opportunity to take this master's program.

Contents

Abstract **i**

Acknowledgments **iii**

1 Introduction **1**
1.1 Problem Statement 3

2 Background and Related Work **5**
2.1 DoS attack definition 5
2.2 DDoS attack techniques 6
2.2.1 Network layer attacks 7
2.2.2 Transport layer attacks 8
2.2.3 Application layer attacks 9
2.2.4 Burst attacks 10
2.3 Yo-Yo attack 10
2.4 Auto-Scaling 11
2.4.1 Auto-Scaling in Cloud 12
2.4.2 Auto-Scaling in Containerized Environment 14
2.4.3 Auto-Scaling in Kubernetes 15
2.5 Related Work 15
2.5.1 Yo-Yo attack in a cloud auto-scaling setting 15
2.5.2 Yo-Yo attack in Kubernetes auto-scaling 17
2.5.3 Yo-Yo attack in container-based environments 17

3 Approach **19**
3.1 Overview of TASD system 19
3.2 TASD modules 20
3.3 Test Environment 23
3.3.1 AWS resources 23
3.3.2 Infrastructure as a Code 26
3.3.3 External tools 27
3.3.4 TASD database 27
3.4 Experiments 27
3.4.1 Experiment 1 28
3.4.2 Experiment 2 28
3.4.3 Experiment 3 28
3.4.4 Experiment 4 29

4 Results **33**
4.1 Experiment 1 . . . 33
4.1.1 DoS attack and auto-scaling . . . 33
4.2 Experiment 2 . . . 35
4.2.1 Yo-Yo attack and auto-scaling . . . 35
4.2.2 Attacker probe response time . . . 35
4.3 Experiment 3 . . . 36
4.3.1 TASD . . . 36
4.4 Experiment 4 . . . 38
4.4.1 An optimization to TASD . . . 38

5 Discussion **41**
5.1 DoS/DDoS attack and cloud auto-scaling . . . 41
5.2 Yo-Yo attack and cloud auto-scaling . . . 41
5.3 Yo-Yo attack and TASD solution . . . 42
5.4 Yo-Yo attack and Cost . . . 43
5.5 Future Work . . . 43

6 Conclusion **45**

Chapter 1

Introduction

During the last decade, the cloud has been growing as a disruptive technology for business, and the wholescale migration of applications and infrastructure to the cloud has started across many industries. Cloud is not only a tool and technology but also another way to deal with the application lifecycle with faster processes and new delivery models. Cloud computing has grown into a vast and complex ecosystem of technologies, products, and services. In 2021, 41% of EU enterprises used cloud computing and 73% of those enterprises used sophisticated cloud services relating to security software applications, hosting enterprise's databases or computing platforms for application development, testing or deployment [33]. The same studies show that the use of cloud computing increased by 5% points compared with 2020. This increment of cloud computing usage needs a framework to prove guidance that is designed to help enterprises create and implement the business and technology strategies that are necessary for their organization to succeed in the cloud [51]. Cloud Adoption Framework leverages cloud providers' experiences and best practices to help digitally transform and accelerate business outcomes through innovative use of the cloud. At the highest level, CAF organizes guidance into six focus areas: business, people, governance, platform, security, and operations [14]. One of the most important pillars of the CAF is security. The CAF security consists of a systematic process for building security and supporting the cloud application development life cycle. [22] [15]. CAF security introduces a three-layer architecture for data security in cloud:
1- Firewall and access control layer
2- Identity management and intrusion prevention layer
3- Encryption and decryption layer

DoS and DDoS attacks are located in the boundary of layer 1 and layer 2 of CAF. DoS and DDoS attacks have become an immense threat to the Internet infrastructure for a decade. Attacks have become commonplace with a wide range of global victims in everything from commercial websites, educational institutions, public chat servers, and government organizations [53] [37]. A detailed analysis from Cloudflare shows that DDoS attacks have been dramatically increasing over the past few years.

The report shows that the number of attacks increased by 50% in Q3 2020 in comparison with Q3 2019 [60]. Another report from Kaspersky researchers observed a massive increase in the number of attacks in Q3 and Q4 2021. The report shows 400% growth in Q4 2021 in comparison with Q4 2020 [43]. According to the Cloud Security Alliance report [45], the number of DDoS attacks on cloud services has also risen over the years. Increasing the number of attacks causes emerging new detection and mitigation solutions as well [26]. In November 2020, Alibaba Cloud Security Team detected the largest resources exhaustion DDoS attack on their cloud platform, with a peak of 5.369 million QPS (Queries Per Second) [8]. Microsoft mitigated upwards of 359,713 DDoS attacks against their Azure Cloud infrastructures during the second half of 2021 [7].

On the other hand, attackers do not surrender. New kinds of DDoS attacks have emerged to exploit cloud anti-DDoS solutions. Bursts attacks, also known as hit-and-run DDoS, is a new kind of DDoS attack, where the attacker launches periodic bursts of traffic overload at random intervals on online targets [32] [69]. Burst attacks have grown generally so that in a comprehensive survey in 2017, half of the participants cited an increase in burst attacks [12]. This points to the increasing sophistication of hackers, in terms of their ability to better leverage large botnets and develop mechanisms that have the ability to evade detection. Botnet malware technologies accelerate burst attacks, due to scheduling and synchronizing features [27].

Enterprises using cloud services mostly benefit from cloud features. Enhanced cloud scalability and elasticity through auto-scaling allow customers to dynamically scale their applications [67]. Incoming traffic is distributed evenly across multiple endpoints, so individual backend services cannot be overwhelmed until the volume of traffic approaches the capacity of the entire network.

Hostile actors adjust their tactics to correspond to the realities posed by the cloud. Yo-Yo attack is a new attack technique against the cloud auto-scaling feature [2]. In this method, attackers send a burst of request traffic to increase the significant load on the cloud server. It causes triggering auto-scaling mechanism, and the server will be scaled up. Therefore, during this confirmation period, the victim's system has deployed resources that far exceed the required amount. Burst traffic will be stopped after scale-out and waiting for the auto-scaling mechanism to scale-in the server. The attacker continues the latter attack procedure and forces the cloud services to scale-out and scale-in continuously [70]. It causes adding extra load to scale-out the services to respond to the fake requests. In effect, the attacker forces the victim to pay for large amounts of resources that are not actually necessary to handle the legitimate workload. Yo-Yo attack can affect any platform using auto-scaling mechanisms such as container-based-environment [25] and Kubernetes platform [27].

1.1 Problem Statement

Auto-scaling automatically adjusts capacity to maintain steady and predictable performance at the lowest possible cost [67]. The Yo-Yo attack explicitly targets auto-scaling functionalities in cloud environments. Although there are numerous studies focused on conventional DoS and DDoS attacks, the number of studies on mitigating such Yo-Yo attacks for the auto-scaling mechanism in the cloud is limited. This leads to the first problem statement that this thesis will explore:

How does a Yo-Yo attack differ from Dos/DDoS attack toward auto-scaling in the cloud?

In this regard, A Trust-based Adversarial Scanner Delaying (TASD) approach has been proposed to mitigate the Yo-Yo attack in the cloud auto-scaling mechanism [70]. This thesis shows the result of the TASD system implementation of a web service running on the AWS cloud platform and leads to the second problem statement that will explore:

To which extent is the TASD solution resilient to Yo-Yo attacks in real cloud services?

Chapter 2

Background and Related Work

To develop an optimized mitigation solution, it is vital to understand the problem domain completely. Although the research topic narrows down to a specific DDoS attack on cloud platforms, it is important to know about different attack techniques. It helps to have a better understanding of both the problem domain and the current solution space to create efficient approaches. This chapter provides information about different DoS and DDoS attack techniques. Moreover, it explores the scope and characteristics of the Yo-Yo attack as well as the cloud auto-scaling feature.

2.1 DoS attack definition

A Denial-of-Service attack (DoS attack) is an intimidating threat that the perpetrator seeks to make a machine or network resource inaccessible to its intended users [28]. DoS attacks accomplish this by flooding the target with traffic or sending the information that triggers a crash [30]. A DoS attack is characterized by using a single computer to launch the attack. In [42], DoS attacks are classified into five categories based on the attack protocol level:

DoS attacks at the network device level might be done due to a bug or weakness in the network device framework or driver. In 2018, a critical vulnerability has been discovered in Cisco's appliances Adaptive Security (ASA) and Firepower. The vulnerability was leveraged DoS condition to attackers [23]. Attackers can also try to exhaust the network device. For example, attackers can make a network device such as a router down by bombarding the device with packets that the router must filter out and store logs.

Protocol implementation at the OS level causes many DoS attacks. Ping of Death attack is an example in this category [49]. It is possible to crash, reboot or either kill a large number of systems by sending a ping of a certain size from a remote machine. In this attack, ICMP echo requests with data size greater than 65536 are sent to the victim. An IP datagram of 65536 bytes is illegal, but possible to fragment the packet. When the fragments

are reassembled at the other end into a complete packet, it overflows the buffer on some systems and causes abnormality.

Application-based attack is another type of DoS attack. Attackers overcome a service or an entire system using bugs in network applications. Network applications examples are such as file transfer, remote file server, etc. A finger bomb is an example of application-based attack.

Attackers can send a massive amount of data to bombard the network bandwidth of a host. The attack method is called data flooding. Amplification attacks, oscillation attacks, and simple flooding are three common types of flooding attacks. Ping flood, also known as ICMP flood, is a data flooding attack example in which an attacker takes down a victim's computer by overwhelming it with ICMP echo requests, also known as pings [57].

Attackers can implement DoS attacks by taking advantage of certain standard protocol features. The most common connection depletion attack is SYN flooding. By repeatedly sending initial connection request (SYN) packets, the attacker is able to overwhelm all available ports on a targeted server machine causing the targeted device to respond to legitimate traffic sluggishly or not at all.

2.2 DDoS attack techniques

An additional sophisticated type of DoS attack is the Distributed Denial of Service (DDoS) attack. DDoS attacks utilize multiple distributed attack resources to exploit computers or other network resources such as cloud services. From a high level, a DDoS attack is like an unexpected traffic jam clogging up the highway, preventing regular traffic from arriving at its destination. Therefore, in its simplest form of DDoS attack, compromised vulnerable devices, also known as zombies, are used to send malformed packets to a target system. The target system cannot handle such packets and it causes the system to stop or reboot. When this occurs, the users' access will be denied to the respective services and resources [29]. DDoS attacks also affect the cloud services including computing elements, memory storage or the network devices [37]. Attackers try to use the cloud provider resources to overflow cloud users. Cloud DDoS attacks can occur internally or externally. An external cloud DDoS attack starts from outside the cloud environment and targets cloud services. On the other hand, internal cloud-based DDoS attacks occur within the cloud system [26]. In both scenarios, attackers can implement different DDoS techniques. This section goes through some popular attack techniques inside of each category.

2.2.1 Network layer attacks

Network layer DDoS attacks target layer 3 in the OSI model. The network layer is responsible to provide cross-network addresses and constructing routing tables for data packets. Moreover, negotiating and ensuring a certain quality of service falls within the remit of the network layer.

Like all DDoS attacks, the goal of a network layer attack is to slow down or crash a program, service, computer, or network, or to fill up capacity so that no one else can receive any service. L3 DDoS attacks typically accomplish it by targeting network devices and protocols.

IP Spoofing attack is a DDoS attack in the network layer. IP Spoofing is a technique to send a package to someone with a wrong return address list. The receiver cannot stop receiving the packages by blocking the sender's IP address, as the return address is easily changed. Also if the receiver wants to respond to the return address, the response package will go somewhere other than to the real sender. The ability to spoof the addresses of packets is a core vulnerability exploited by many network layer DDoS attacks such as NTP and DNS amplification.

An NTP amplification attack is a DDoS attack in which an attacker exploits a Network Time Protocol (NTP) server [54]. Attackers send a large number of fake requests using a technique called NTP reflection with spoofed source addresses to overwhelm the NTP server. The size of the request can reach up to 400 Gbps [37].

A DNS amplification attack is a type of DDoS attack in which an attacker leverages the functionality of open DNS resolvers in order to overwhelm a target server or network with an amplified amount of traffic, rendering the server, and its surrounding infrastructure inaccessible. Attackers send queries with spoofed IP addresses to the target servers. Then It will send responses that are many times bigger than the victim's clients or servers. For example, as a DNS query consists of approximately 36 bytes, a response message could easily triple that size [41]. As a result, the target receives an amplification of the attacker's initial traffic, and their network becomes clogged with the spurious traffic, causing a DDoS.

A smurf attack is a DDoS attack in that attackers try to flood a targeted server with the Internet Control Message Protocol (ICMP) packets. Network devices, such as routers, use ICMP to send error messages and connect some information. The information indicates success or failure when communicating with another IP address by sending an ICMP echo reply message back to the sender [40]. In smurf-based attacks, spoofed ICMP echo requests are sent to a network device that is configured to relay ICMP messages to all devices behind it. When all of the devices receive the ICMP echo request, the devices reply with an ICMP reply message to the victim host, efficiently amplifying the attack [46].

2.2.2 Transport layer attacks

OSI Model Layer 4, or the transport layer, provides transparent transfer of data between end systems, or hosts, and it is responsible for end-to-end error detection, recovery, and flow control. It ensures complete data transfer, which treats data as a stream of bytes through the TCP or UDP. The attackers manipulate the communication protocols in the transport layer to execute DDoS attacks. SYN Flood is a common technique to misuse protocols. SYN Flood means sending a request to a worker. The worker receives the request and provides the reply, and waits for confirmation before replying to the next request. Then the worker receives many more requests without any confirmation until the worker cannot handle all the requests and become overwhelmed [66].

TCP SYN Flood attacks exploit the weakness of the three-way handshake in TCP. The client starting a connection sends an initial SYN request to a server. The server sends back the SYN/ACK packet and waits for the client to confirm it by sending the final ACK packet [30]. According to the SYN Flooding method, attackers send a large number of the SYN packets to the target and never send ACK replies [19]. Hence, the final ACK message will never be sent to the victim server system and due to buffer queue limitation for new connections, the server cannot reply to other valid incoming requests, and it gets overwhelmed. Attackers can use different spoofed source IP addresses or even use a direct attack with their valid IP address [20].

UDP Flood attack is a type of transport-based layer DDoS attack. A UDP Flood misuses the steps that a server takes to respond to a UDP packet under normal conditions. The server initially checks whether any programs are running on the specific UDP port. If a program listens on that UDP port, the request sends to the program, else the server responds with an ICMP packet to inform the sender that there is no program to listen [68]. In the same way, attackers send a large number of UDP packets to a target's random or specified ports. The target server checks the applications list to identify the port consumer. As the port is not used by any application, the server replies with a message "destination unreachable" using ICMP packet. The ICMP packets are sent to the spoofed IP address. As a result of the targeted server utilization and responding to each received UDP packet, the target's resources can become quickly exhausted when a large flood of UDP packets is received. It results in denial-of-service to normal traffic [29].

If the attacker sends a bunch of spoofed UDP packets with the source address of another victim, it causes UDP storm attack [47]. The attacker sends UDP packets to echo port of the first victim as if it is coming from the second victim in the same network. First victim sends a reply with some

data to echo port of second victim as it does not know the spoofed address. Then the second victim replies with data to the first victim, and this loop goes on until there is external interference.

2.2.3 Application layer attacks

The application layer in the OSI model is the closet layer to the end system user. In this layer, communication from one-user to another begins by using the interaction between the application layer. The layer provides the connection to the lower layers. Attackers may target the application itself by using application-layer attack. The main goal of the application-based attack is to take out an application or an online service using vulnerabilities or problems in application protocol [64]. These application-layer attacks, in contrast to network and transport layers, attack, are particularly difficult to detect and mitigate. Attackers use very low request volumes that generate only a small volume of network traffic [17]. The following section represents some application-based attack techniques.

HTTP flooding attack is the most common application layer attack [48]. HTTP flood is a complex attack because it does not use malformed packets, spoofing, or reflecting techniques to execute an attack. Attackers exploit the legitimate HTTP GET and POST requests, therefore the attack needs less bandwidth than other attacks to affect the targets. The attack is most effective when it forces the server or application to allocate the maximum resources possible in response to every single request. Thus, the perpetrator generally aims to inundate the server or application with multiple requests that each is as processing-intensive as possible. Therefore, the HTTP flood attacks using POST requests tend to be the most resource-effective from the attacker's perspective; as POST requests may include parameters that trigger complex server-side processing. On the other hand, HTTP GET-based attacks are simpler to create and can more effectively scale in a botnet scenario.

HTTP flood attacks are difficult to detect from normal traffic because they use standard HTTP requests. So, the normal IDS techniques that use signature-based detection are not able to detect HTTP flooding attacks. The most highly-effective mitigation mechanism relies on normal users' browser behavior before detecting abnormal browser patterns [71]. For example, Yatagai et al. proposed two solutions to detect HTTP flooding attacks; firstly, by monitoring the correlation between the time a user consumes to browse a web page and the size of that web page. Compromised hosts can therefore be removed as it is expected that they do not follow the normal browsing pattern behavior. The second solution focuses on the user browsing behavior and history [71].

XML flooding attacks can be categorized as application-layer attacks. A SOAP message works with HTTP and is an XML document, which consists of a SOAP envelope, an optional SOAP header, and a SOAP body. X-

DoS, an Extensible Markup Language DoS attack is a content-borne DoS attack that occurs when an XML message is sent with a multitude of digital signatures, and a naive parser looks at each signature. It causes all the CPU resources to be used and the service becomes down. X-DoS can be carried out using less sophisticated tools due to its ease of implementation [56].

2.2.4 Burst attacks

Burst attacks are gaining favor among attackers because they enable perpetrators to attack multiple targets, one after the other, with short and high-volume traffic bursts, in a rapidly repeating cycle. The burst attacks have been increasing in recent years. In a survey published by Cisco in 2017, it has been found that 41 percent of organizations suffered from burst DDoS attacks [16]. The same report shows that burst tactics are usually aimed at gaming websites and service providers due to their clients' sensitivity to service availability and their inability to sustain such attack maneuvers. Burst attacks have also known as hit-and-run DDoS because the solution uses repeated short bursts of high-volume attacks at random intervals. As an example, each short burst can last only a few seconds. In 2017, a top-five U.S. carrier witnessed a tenfold increase in burst attacks, where 70% to 80% of attacks were less than a minute long [18].

Detecting a burst attack is not difficult but the rate of bad and legitimate traffic to certain threshold results in a high level of false positives in the detection. Thus, to minimize the false positive, it needs to create a signature to only block the attack traffic. If the attack vector changes across bursts, the signature needs to adapt to the changing attack characteristics. The process of repeated manual signature adjustments can become a labor-intensive task, which renders the whole protection strategy infeasible.

Machine learning techniques help to detect and mitigate burst attacks as a Behavioral DoS (BDoS) protection technology. BDoS protection method uses different features on network protocols to learn and compare real-time statistics with the learned baseline. A fuzzy-logic inference system measures the degree-of-attack (DoA) surface. BDoS considers an attack to have started and triggers attack handling only when the overall DoA surface for the combined parameters is high. This guarantees accurate detection of attacks [18].

2.3 Yo-Yo attack

The Yo-To attack tries to leverage the strength of the cloud against an organization that uses it. Specifically, the attack attempts to abuse auto-scaling mechanism in the cloud [70][2]. The attacker sends a burst of request traffic to a target running on the cloud. It causes increasing the significant load on the cloud. Auto-scaling mechanism tries to scale-out the resources to mitigate the high traffic load. The attacker recognizes the scaling up by

measuring probe packets' response time [62][2], then stops sending traffic and waiting for auto-scaling mechanism to scale-in the resources. Turning off the DDoS attack in scale-in stage is a key point of the Yo-Yo attack. According to the autoscaler functionality, resources are not being scaled in immediately [1]. The attacker starts sending the burst traffic to trigger the auto-scaling mechanism to scale-out again and the cycle repeats over and over.

The Yo-Yo attack goal is not to take the services offline necessarily. The main goal of the attack is to imply financial damage. Although cloud providers offer different pricing models, cloud service providers operate on a consumption-based model. This means that end-users only pay for the resources that they use. The consumption-based model brings some benefits such as the ability to pay for additional resources when they are needed and also the ability to stop paying for resources that are no longer needed. Therefore, as the victim's system scales in resources during each cycle, significant charges accrue to the victim's account. In effect, the attacker forces the victim to pay for large amounts of resources that are not actually necessary to handle the current legitimate workload. For instance, AWS charges the usage of EC2 only for what is used. The cost is the price per instance-hour consumed for each instance, from the time an instance is launched until it is terminated or stopped. Each partial instance-hour consumed will be billed per second for Linux, Windows, Windows with SQL Enterprise, Windows with SQL Standard, and Windows with SQL Web Instances, and as a full hour for all other instance types [11]. It means that after scaling in an instance, it costs the service provider price per second, while the scale-in happened through a Yo-Yo attack.

2.4 Auto-Scaling

Elasticity is the ability to automatically or dynamically increase or decrease resources as needed. Elastic resources match the current needs, and resources are added or removed automatically to meet future needs. Automatic scaling is the key difference between scalability and elasticity. Auto-scaling is a cloud computing technique based on elasticity features. It enables organizations to scale cloud services such as server capacities or virtual machines up or down automatically, based on defined situations such as traffic or utilization levels. According to [67], the main purpose of the auto-scaling mechanism is to cope with changes in the traffic load. Auto-scaling can use to cope with both predictable and unpredictable changes due to an abnormality in service features or flash crowds or even malicious DDoS attacks. It means that auto-scaling can mitigate DDoS attacks by increasing the number of instances as much as needed. This solution, however, comes with an economic penalty, termed Economic Denial of Sustainability attacks (EDoS).

In November 2008, Hoff et al.[21] firstly hypothesized the presence of a novel strain of the denial of service threat, which was termed Economic Denial of Sustainability (EDoS). It described a specific family of attacks against the different cloud computing platforms, where the intruder aimed on increasing the economic costs derived from both maintenance and provision of the services offered, hence making their support less viable, even achieving denial.

Therefore, the victim of DDoS attack on a system running in an auto-scaling feature needs to pay for extra resources to process the fake traffic requests, and it provides no real benefit to the victim. However, auto-scaling is regarded as a good enough solution, since it ensures that the service will continue to run with good performance. The below sections presents different auto-scaling mechanism.

2.4.1 Auto-Scaling in Cloud

In cloud computing, scaling is the process of adding or removing computation, storage, and network services. The scaling helps to meet the demands a workload makes for resources in order to maintain availability and performance as utilization increases. Scaling generally refers to adding or reducing the number of active servers, known as instances in the cloud, being leveraged against the client workload's resource demands. Scaling can be horizontally or vertically. Horizontal scaling known as scale-out, simply adds more instances to the existing instances. Vertical scaling or scale-up adds power to the existing machine infrastructure by increasing power from CPU or RAM to existing machines.

Cloud providers have implemented different auto-scaling mechanisms for different cloud services. Auto-scaling mechanism can detect an unhealthy instance to terminate it, and launch another instance to replace it. Moreover, it helps ensure that the application always has the right amount of capacity to handle the current traffic demand. Dynamic scaling provides better cost management for cloud users because they pay for the instances they use, they save money by launching instances when they are needed and terminating them when they are not. Each cloud solution comes with its own auto-scaling engine. The below sections introduce some auto-scaling mechanisms provided by cloud providers. Basically, in each of these systems, the underlying algorithms allow the cloud customer to define scaling criteria and the corresponding thresholds for overload and underload.

Amazon EC2 auto-scaling is the autoscaler resource provided by AWS. EC2 auto-scaling helps clients to ensure that the correct number of Amazon EC2 instances is available to handle the load. It can launch or terminate instances as demand on application load increases or decreases. AWS auto-scaling supports different scaling policies. Manual scaling updates

the desired capacity of the auto-scaling group or updates the instances that are attached to the auto-scaling group [50]. Dynamic scaling creates target tracking scaling policies for the resources in your scaling plan based on a specific metric. These scaling policies adjust resource capacity in response to live changes in resource utilization. The intention is to provide enough capacity to maintain utilization at the target value specified by the scaling strategy [31]. Amazon auto-scaling supports the following types of dynamic scaling policies:

1- Simple scaling - scale-out and scale-in based on a single scaling adjustment.

2- Step scaling - Increase or decrease the current capacity of the auto-scaling group based on a set of scaling adjustments that vary based on the size of the alarm breach.

3- Target tracking scaling - Scaling based on a target value for a specific metric.

AWS introduced an auto-scaling policy using a machine learning algorithm to predict scaling. The predictive scaling analyzes the resource's historical workload and forecasts the future load. Load forecasting is a key concept of predictive scaling. AWS auto-scaling analyzes up to 14 days of history for a specified load metric and forecasts the future demand for the next two days [58].

Microsoft Azure provides built-in auto-scaling for most computation options. Azure virtual machines scale sets can scale in and out VMs independently. Azure app services and cloud services have a built-in auto-scaling mechanism. Moreover, Azure supports a custom auto-scaling solution, which can define custom rules based on these metrics, and use Resource Manager REST APIs to trigger auto-scaling. Azure monitor auto-scale provides a set of policies for VM scale sets. It scales up the resource when load increases to ensure availability. Similarly, at times of low usage, scale-in, so cloud users can optimize cost [52].

Google Cloud Platform offers an auto-scaling mechanism for VM from a Managed Instance Group (MIG). Autoscaler can use one or more signals to scale the group via auto-scaling policy. GCP auto-scaling policy supports both target utilization metrics and schedules. If the workloads running on the GCP vary predictably with daily or weekly cycles, predictive auto-scaling mode is recommended by Google. An auto-scaling mode called "scale-in controls" is introduced for workloads that take many times to initialize. It reduces the risk of response latency caused by abrupt scale-in events by configuring scale-in controls. If the load spikes are expected to follow after declines, the scale in rate can be limited to prevent auto-scaling from reducing a MIG's size by more VM than that of the workload can tolerate [34].

Other cloud platforms provide auto-scaling mechanisms for their cloud services. Heat is the Openstack auto-scaling component [55]. The auto scaler for IBM Cloud Virtual Servers provides the ability to automate the manual scaling process that is associated with adding or removing instances to support services running on their cloud platforms [39].

The auto-scaling feature is not limited to IaaS (Infrastructure as a Service) solutions. Pivotal Cloud Foundry is a cloud PaaS (Platform as a Service) to provide fast application development and deployment environment. App Autoscaler is a PCF service that scales apps based on performance metrics or a schedule. App Autoscaler adjusts app instance counts based on metrics threshold. The maximum and the minimum number of instances for an app can be modified manually or using a scheduler. Every 35 seconds, App Autoscaler averages the values of a given metric for the most recent 120 seconds and makes a scaling decision. This controls the cost of running apps while the app performance is maintained [3].

2.4.2 Auto-Scaling in Containerized Environment

Containerization is one of the most efficient methods of virtualization in the development environment. It provides portability, which ensures that the container works the same way regardless of where to deploy. Containers improve efficiency by optimizing resource usage and minimizing overhead. Cloud providers also offer container services. For example, AWS Elastic Container Service is a scalable container management service that can manage Docker containers on a cluster of instances. AWS ECS container runs as a task. Automatic scaling is the ability to increase or decrease the desired count of tasks in the ECS service automatically. AWS CloudWatch metrics can be used to scale-out the service to deal with high demand at peak times, and also scale-in service to reduce costs during periods of low utilization [61]. The same services are offered by other cloud providers. For example, Microsoft Azure introduces a scaling rule in Azure Container Apps.

In contrast to cloud Container services, Docker does not have a built-in auto-scaling mechanism. Docker is one of the most common containerized technologies, and Docker swarm is a built-in container orchestration tool. Docker Swarm can manage the number of desired tasks to run. When services scale-out or scale-in, the swarm manager is automatically adapted by adding or removing tasks to maintain the desired state. The swarm manager node constantly monitors the cluster state and reconciles any differences between the actual state and the expressed desired state. But it cannot scale-out or scale-in based on the load on the services or any specific metric. Therefore, other modern container orchestration solutions such as Kubernetes are useful [65].

2.4.3 Auto-Scaling in Kubernetes

Container-based microservices architectures have profoundly changed the way development and operations teams test and deploy modern software. Large and small software companies are now alike deploying thousands of container instances daily, and that is a complexity of scale they have to manage. Kubernetes is an open-source container orchestration platform, originally developed by Google, designed to automate the deployment, scaling, and management of containerized applications. In the Kubernetes technology Pod is a computing unit that can host one or more containers. The containers can share resources such as storage and network. This means that a Pod can run a single container, but it can also run several ones that need to work together.

Kubernetes has a Horizontal Pod Autoscaler feature that can update a workload resource automatically. The main aim of the Horizontal Pod Autoscaler is to scale the workload to match demand. Kubernetes implements horizontal pod auto-scaling as a control loop that runs intermittently. The default interval is 15 seconds and can be set by the Kubernetes manifest configuration. Once during each period, the controller manager queries the resource utilization against the metrics specified in each Horizontal Pod Autoscaler definition.

The Horizontal Pod AutoScaler has a controller module that operates on the ratio between desired metric value and current metric value:

$$desiredReplicas = ceil[currentReplicas * (currentMetricValue/desiredMetricValue)$$

Kubernetes Horizontal Pod AutoScaler controller module calculates the *desiredReplicas* and checks the tolerance to decide on the final values. The control plane also considers whether any metrics are missing, and how many Pods are Ready. All Pods with a deletion timestamp set are ignored, and all failed Pods are discarded. Kubernetes Horizontal Pod AutoScaler policy supports custom metrics as well as multiple metrics to scale on [38].

2.5 Related Work

The Yo-Yo attack is a new DDoS technique, and it has emerged due to the cloud auto-scaling feature. Therefore, the number of research is limited. This section presents some relevant research that can be used to understand Yo-Yo attack aspects and possible mitigation solutions.

2.5.1 Yo-Yo attack in a cloud auto-scaling setting

The Yo-Yo attack which is described in [2], is the first work on auto-scaling mechanism vulnerability. There is a discussion that the Yo-Yo at-

tack and auto-scaling mechanism cause the Economic Daniel of Sustainability (EDoS) attacks [63][9]. This paper shows that rather than the economic effect of the attack, the Yo-Yo attack can cause substantial performance damage. During the repetitive scale-out process, which takes several minutes due to the instance startup process [35], the cloud service suffers from substantial performance degradation. The article shows that auto-scaling policy configuration is an important factor to minimize the effect of the Yo-Yo attack. So, the Yo-Yo attack can also be a type of Reduction of Quality (RoQ) attack [36].

The article formulated the Yo-Yo attack damage. The calculation considered a Yo-Yo attack as n cycles with the duration T, on-attack and off-attack periods of t_{on} and t_{off} respectively. The power of attack assumed in a steady state with k value, so the attack duration would be defined:
$t_{on} = I_{up} + W_{up}$
$t_{off} = I_{down} + W_{down}$
$\mathrm{T}=\mathrm{t}_{on} + t_{off}$

where I_{up} and I_{down} show the scale-out and down duration, and W_{up} and W_{down} represent the instance warm up and warm down duration. The article also defines a formula to calculate the performance and economic damage $D^{attack}(k)$ by a Yo-Yo attack with a power of k. The Yo-Yo attack performance damage can be calculated via the below formula:

$$D_p^{YoYo} = k \times (\mathrm{I}_{up} + W_{up})/T$$

And the performance damage can be estimated as follows, where m is the instance number:

$$D_e^{YoYo} = k \times \mathrm{m} \times (\mathrm{T}^a - I_{up})/T$$

Based on this study, [70] proposed a detection and mitigation system for a Yo-Yo attack in cloud auto-scaling mechanism. A Trust-based Adversarial Scanner Delaying (TASD) approach has been suggested. The TASD approach is inspired by two key factors. First, compared with benign users, the Yo-Yo attacker may initiate burst requests and cause the auto-scaling mechanism to scale-out more frequently. Moreover, the attacker shows a large difference in request load between the scale-out and scale-in phases. Therefore, the TASD system assigns a trust value to each client that can be updated based on their behavior. The trust value is a Quality of Service (QoS) for each client. So, under the QoS constraints, TASD injects certain delays to suspicious requests in order to manipulate the response time and deceive the attackers.

The article shows that TASD can largely decrease the number of scaling as well as Yo-Yo attacks. The experiments show that using the TASD system resulted in a decrease of 38% of attacks. Also, TASD can reduce the number of scaling and leads to a 41% decrease in the number of scale-

outs, compared with the non-defense environment. The experiment results show that the interval of auto-scaling increases significantly in TASD. It shows that the scaling interval increased from 22.22s in a non-TASD system to 33.34s in their TASD experiment.

Implementation of the TASD system in a real cloud environment is suggested as future work. The first part of this master thesis is an effort to implement the TASD prototype in a real web service running on the AWS cloud.

2.5.2 Yo-Yo attack in Kubernetes auto-scaling

To continue studying the Yo-Yo attack and auto-scaling, [27] did research on the Yo-Yo attack and Kubernetes auto-scaling mechanism. The article compares the Yo-Yo attack effect on Virtual Machines and containerized environments. The results try to emphasize that Kubernetes has better performance resilience than VM against Yo-Yo attacks, but shares a similar vulnerability to economic damage. It represents that performance degrading by Yo-Yo attack on Kubernetes is significantly lower than of VM with almost no packet errors. Kubernetes can spin up a pod very quickly to respond to the increased traffic. It means that $I_u p$ and $W_u p$ are significantly smaller than values in VM. On the other hand, the Yo-Yo VM attack results in almost the same relative economic damage as the Yo-Yo attack causes a Kubernetes cluster.

The article proposed an ML method to detect Yo-Yo attacks in Kubernetes. Time series data generated by a sample Kubernetes cluster has been used as a primary foundation of the experiments. The incoming traffic has been labeled in two classes $Attack(1)$ and $Regular(0)$. XGBoost classification algorithm has been used to detect the Yo-Yo attack in Kubernetes. The system has been trained by observing the auto-scaling behavior under attack and normal legitimated traffic, and the experimental data has been collected from 21 samples. The article applies different machine learning algorithms, but the result shows that XGBoost algorithm has high accuracy and a lower false positive rate.

2.5.3 Yo-Yo attack in container-based environments

Viktor Danielsen at his master thesis [25] investigated the Yo-Yo attack in a containerized environment. The article shows the result of a Yo-Yo attacking a Dockerize web server running inside a custom auto-scaling mechanism. A script has been developed to read statistics from a load balancer and apply scale Docker containers accordingly.

A detection method has been introduced based on adversarial requests. It compares the number of occurrences of a client IP address related to the

scale-out and scale-in phases, and simply considers the IP malicious if the number of requests in scale-out is more than the number of requests in scale-in.

Chapter 3

Approach

The project is an implementation of the TASD prototype solution suggested in [70]. This section introduces the overview of the Yo-Yo attack detection approach and describes the mitigation solution in detail.

3.1 Overview of TASD system

Figure 3.1 depicts an overview of the TASD system architecture to detect and mitigate the Yo-Yo attack. The load balancer acts as the "traffic cop" sitting in front of the auto-scaling and routing client requests across all servers capable of fulfilling those requests. It helps in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance. When a new server is added to the auto-scaling group, the load balancer automatically starts to send requests to it, and in case of scaling-in, the load balancer redirects traffic to the remaining online servers. The TASD architecture has been designed as a distributed system and runs as a "side-car". It means that TASD service runs on each instance where the web application is running. Therefore, when the system scales-out and a new instance is added to the auto-scaling group, TASD distributed service can work as a defense mechanism, independently.

TASD system has two main functions: detection and mitigation. The attack detection module firstly determines the type of request for each user. If it is identified as suspicious, this request will be forwarded to the second auto-scaling group. The instances inside the second auto-scaling group run a web service, which replies to the requests with a two-second delay. If a request identifies as malicious, the request will be dropped directly. Otherwise, it will be marked as normal and forwarded to the main auto-scaling module. The next section discusses the TASD modules in more detail.

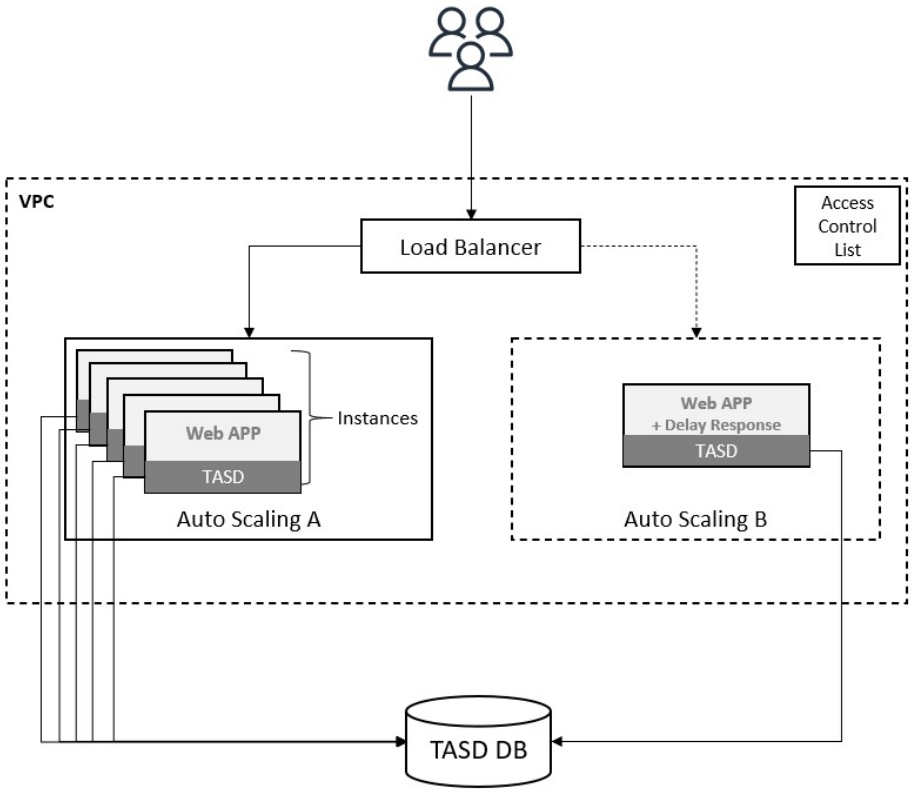

Figure 3.1: The project architecture diagram using TASD system as a "side-car" inside each instances of auto-scaling group.

3.2 TASD modules

According to the characteristics of the Yo-Yo attack, the attacker frequently triggers the auto-scaling mechanism. The attacker continuously sends burst traffic to increase the load of the system. It causes scaling out the instances. Then attacker stops sending traffic and waits for scale-in. Subsequently, compared with benign traffic, the requests from Yo-Yo attackers frequently occur during the server overload. This feature shows that there is a large difference in load volumes between scale-out and scale-in phases. TASD system can detect the attack by comparing the request numbers and cloud auto-scaling status. The detection module adds a Quality of Service (QoS) number which is known as a trust value for each user. Therefore, the trust value is selected based on the below indexes:

- Status of the auto-scaling.
- The difference of load volumes between scale-out and scale-in phases.

The detection module is responsible to capture received packages and maintaining a list of trust values for all users. Firstly it checks whether the

user exists in the trust value database. If not, it adds a new entry for the user IP address and initial trust value (T_{init}). The module checks the auto-scaling status and counts the request number of each user during scale-out process. In addition, in each scale-out process, the detection module records k users that have the most requests load volume in the trust value list. The user's IP address and the number of requests are stored in an array called S. Then after scaling-in, the request number of k user in the array S compares with the current request number, if the request is loaded down by M, the attack detection module decreases the user's trust value by one. Algorithm 1 shows the Pseudo code for assigning trust value by TASD. Then, based on the trust value, the module marks different requests as normal, malicious, or suspicious.

Algorithm 1 The Pseudo code for assigning trust value by TASD

```
for each request r do
    if the user u in request r is not in trust value db then
        Add user u with default trust value T_init to trust value db;
    end if
    if Auto-scaling in Scale-out then
        N_i ← number of requests for each users;
        {S} ← k users with top t request numbers;
    end if
    if Auto-scaling in Scale-in then
        for each user u_i in {S} do
            N'_i ← number of requests for u_i;
            if N' - N > M then
                Update trust value of user u_i with T_i = T_i -1 ;
            end if
        end for
    end if
end for
```

The mitigation module is responsible for forwarding or dropping the packets. For each received request, The TASD mitigation module checks the trust value number of each user. As shown in the algorithm 2, if the trust value is lower than $T(suspicious)$, then the TASD adds a new rule to the load balancer to forward all the requests from the specific source IP address toward the second auto-scaling group. Moreover, a user can be denied access by blocking its IP address in the Network Access Control List, if the trust value is lower than $T(malicious)$.

Considering a situation that which a benign user has a suspicious behavior in a period, then the trust value assigned to the user will be decreased wrongly, and it may cause degradation in the quality of service. This is a bottleneck in the mitigation mechanism in the suggested algorithm 2 and may cause false-positive mitigation. To optimize the algorithm, TASD either calculates the duration of request forwarding $t_{forward}$ or denies t_{block}

Algorithm 2 The Pseudo code to mitigate attack by TASD

```
for each request r do
    if the trust value of request T_i < T_suspicious then
        Add user u to ALB forwarding rule;
    end if
    if the trust value of request T_i < T_malicious then
        Add deny rule for user u in VPC ACL;
    end if
end for
```

rules. So, it keeps rules for a specific period of $t_{release}$ and then increases the trust value by one. As a result, if the user shows suspicious/malicious behavior, the trust value will be decreased. Algorithm 3 describes the Pseudo code to optimize mitigation algorithm.

Algorithm 3 The Pseudo code to mitigate attack with dynamic trust value by TASD

```
for each request r do
    if the trust value of request T_i < T_suspicious then
        if t_forward for user u_i > t_release then
            Update trust value of user u_i with T_i = T_i +1 ;
        else
            Add user s to ALB forwarding rule;
        end if
        if the trust value of request T_i < T_malicious then
            if t_block for user u_i > t_release then
                Update trust value of user u_i with T_i = T_i +1 ;
            else
                Add deny rule for user u_i in VPC ACL;
            end if
```

The algorithms have been developed with Python language. It has a distributed architecture so the TASD services running on each instance needs access to a common database. Table 3.1 lists all tables of TASD database:

Table 3.1: TASD database tables' detail

Table name	description
user-trust-value	List of users and assigned trust value
scaling-status	The latest auto-scaling status
top-users	k users with top t request numbers
suspicious-users	List of users with suspicious flag
malicious-users	List of users with malicious flag

The idea behind the experiments shown in this thesis is to test how the proposed algorithms are practical, as opposed to testing them theoretically. This means that the tests will be run like a real business would run a Yo-Yo attack detection system in a customer-facing cloud platform. To evaluate the Yo-Yo attack effect and cloud auto-scaling feature, different scenarios will be implemented. The experiments will start with a normal DoS attack on a web service running on AWS cloud with auto-scaling. Then, a Yo-Yo attack will be simulated on the same web service. Finally, the mitigation solution will be applied to see how it can help to detect and prevent the Yo-Yo attack.

3.3 Test Environment

3.3.1 AWS resources

The AWS cloud platform is used to test and simulate a Yo-Yo attack. A simple web application has been deployed on AWS Elastic Compute Cloud (EC2) managed by AWS Auto Scaling group. An AWS Auto scaling group consists of a collection of Amazon EC2 instances as a logical group for the purposes of automatic scaling and management. It also provides more features such as health check replacements and different scaling policies. To distribute the incoming traffic across multiple EC2 instances, AWS Application Elastic load balancer is used. It monitors the health of its registered targets and routes traffic only to the healthy targets.

The EC2 instances in an auto-scaling group have a path, or lifecycle, that differs from that of other EC2 instances. The lifecycle starts when the auto-scaling group launches an instance and puts it into service. The lifecycle ends when the auto-scaling group takes the instance out of service and terminates it. The Figure 3.2 shows the transitions between instance states in the Amazon EC2 Auto Scaling lifecycle.

To evaluate and compare the DoS and Yo-Yo attacks, an AWS lunch template without any detection system is created. The launch template is an EC2 instance configuration template that AWS Auto scaling group uses to launch EC2 instances. It can include the Amazon Machine Image (AMI) ID, instance type, security group configuration, and user data script. A custom AMI has been created to implement the non-TASD project. Amazon Linux 2 AMI (HVM) – Kernel 5 is the base image of the AMI. Figure 3.4 illustrates the service high-level design of the AMI without any Yo-Yo attack detection system. It contains a flask web application responding to the requests behind an Nginx web server. To Forward web server requests to the Flask web application, the Gunicorn WSGI middleware has been configured. The acronym "WSGI" stands for Web Server Gateway Interface, which is an esoteric way of saying "how a webserver communicates with Python." WSGI and its predecessors are a form of middleware for webservers like Nginx to serve Python apps. To provide a robust system, all the services have been

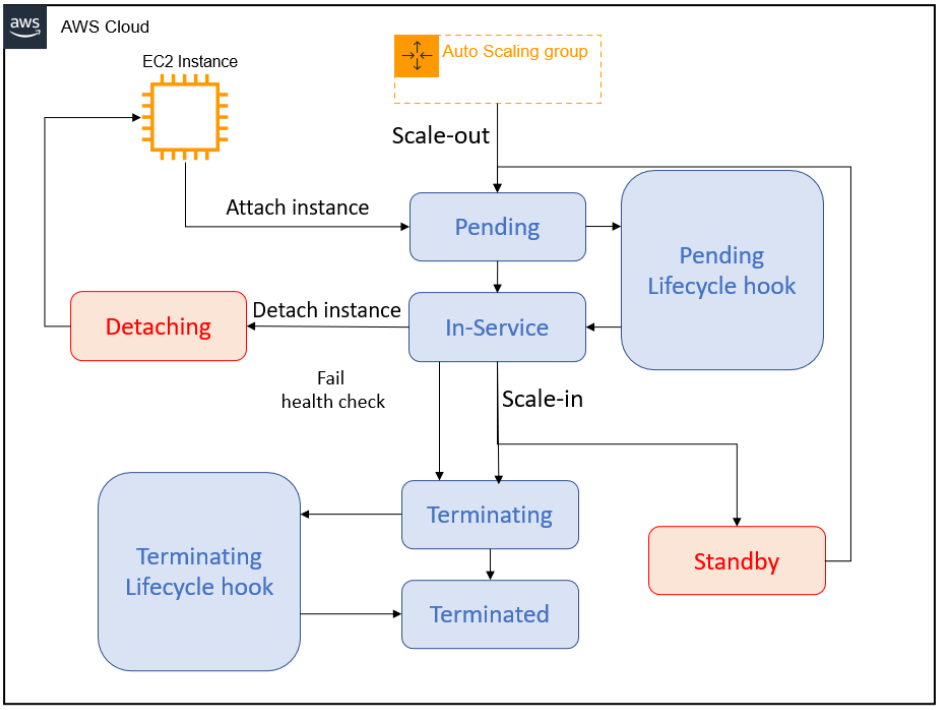

Figure 3.2: The transitions between instance states in the Amazon EC2 Auto Scaling lifecycle.

configured and run as systemd service. So, after spinning a new instance, the web server will be up and running.

AWS provides a wide range of instance types for different usages [10]. Amazon EC2 T2 instance has been used in the project. T2 instances are Burstable Performance Instances that provide a baseline level of CPU performance with the ability to burst above the baseline. The main reason to select T2 instance type is the balance of computation, memory, and network resources. Moreover, the T2 instances are low-cost general purpose instance types, and eligible for Free Tier.

Amazon EC2 security group works as a virtual firewall to the EC2 instances to control incoming and outgoing traffic. Inbound rules control the incoming traffic to the instance, and outbound rules control the outgoing traffic from the instance. When an instance is launched, VPC assigns the default security group to it. A default security group is named "default", and it has an ID assigned by AWS. By default, it allows inbound traffic from network interfaces and instances that are assigned to the same security group and allows all outbound IPv4/IPV6 traffic. TASD project uses a custom security group. Tables 3.2 and 3.3 show the security group details. HTTPS has not been configured as a simplicity of implementation.

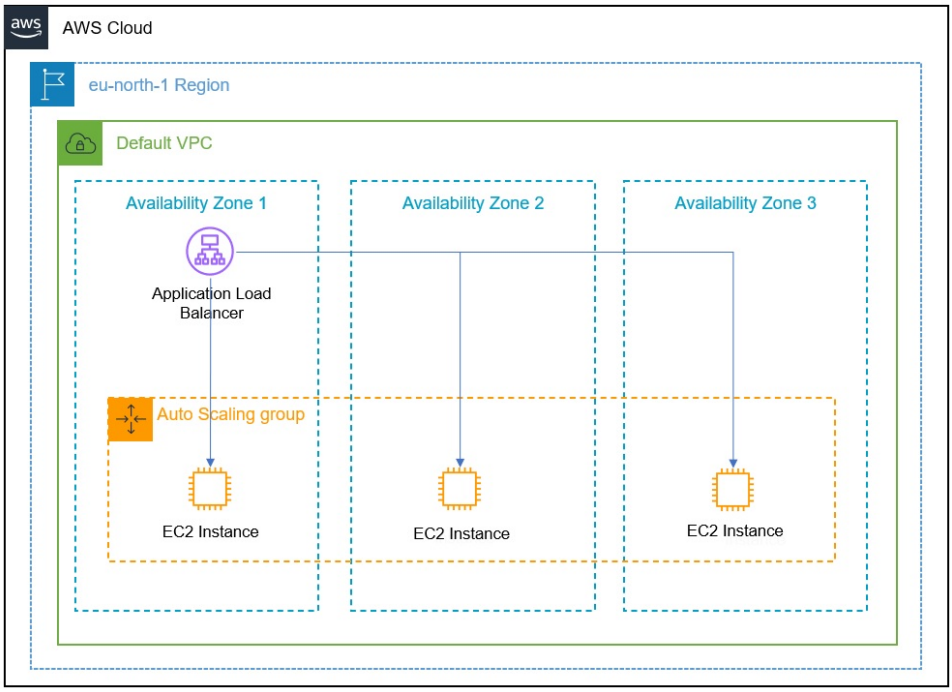

Figure 3.3: The project architecture diagram of the deployment of non-TASD system on AWS cloud.

Table 3.2: Security group inbound rules

Source	Protocol	Port number	description
0.0.0.0/0	HTTP	80	http port to access web application
0.0.0.0/0	SSH	22	ssh port to connect to instance

The AWS Auto scaling has been used to activate the auto-scaling feature. It has been configured with the desired capacity of one instance with a minimum instance of one and a maximum of five. The desired capacity shows the number of instances that the service starts with. In case of any increasing load that needs scale-out, the number of instances can be reached up to five. After scaling-out or scaling-in instances, auto-scaling waits for a cooldown period to end before any further scaling activities initiated by simple scaling policies can start. The intention of the cooldown period is to prevent the auto-scaling group from launching or terminating additional instances before the effects of previous activities are visible. Therefore, to have a sharp scaling, the default cooldown was set to a minimum value equal to 30 seconds.

According to section 2.4.1, AWS provides different scaling strategies. A simple scaling policy for Amazon EC2 auto-scaling has been configured in the project's experiments. With the simple scaling, scaling metrics and threshold values for the metrics monitoring alarms that invoke the scaling

Figure 3.4: Services high level design running on EC2 instance; Nginx web server, Gunicorn WSGI and sample Flask web application.

Table 3.3: Security group outbound rule

Source	Protocol	Port number	description
0.0.0.0/0	All	All	Allows all outbound IPv4 traffic.

process can be defined. Also, it is possible to define how the AWS Auto scaling should be scaled when a threshold is in breach for a specified number of evaluation periods. The AWS CloudWatch is used to capture all metrics and trigger the AWS Auto scaling to scale-out or scale-in the instances.

Elastic Load Balancer publishes data points to Amazon CloudWatch. CloudWatch retrieves statistics about those data points as an ordered set of time-series data, known as metrics. Think of a metric as a variable to monitor, and the data points as the values of that variable over time. The metric to trigger the autoscaler is set to ELB RequestCount. The RequestCount metric is the number of requests processed over IPv4 and IPv6. This metric is only incremented for requests where the load balancer node was able to choose a target. For example, HTTP 460, HTTP 400, and some kinds of HTTP 503 and 500 are not reflected in this metric. The metrics are checked with the evaluation period of one which is a number of periods over which data is compared to the specified threshold in 60 seconds.

The scale-out policy is triggered if the sum of the Request Count exceeds over 1K and the scale-in policy is triggered by the sum of the Request Count goes less than 100. The threshold values have been selected based on the Flask application and Gunicorn WSGI workers.

3.3.2 Infrastructure as a Code

To make the implementation fast and consistence, Hashicorp Terraform has been used as an Infrastructure as a Code solution. It helps to define the cloud resources in human-readable configuration files. Terraform AWS provider is used to interacting with the AWS resources. The project uses

Terraform version 1.1.9 and Terraform AWS provider version 3.75.1.

3.3.3 External tools

To simulate a burst attack, the Locust tool has been used. Locust is an open-source tool to run a load test on web applications. It supports running load tests distributed over multiple machines, and can therefore be used to simulate millions of simultaneous users [13].

A Python script has been developed to send normal traffic as a benign user. The script uses *requests* module to send GET requests toward the load balancer DNS. It sends random requests in the interval of one to three seconds.

3.3.4 TASD database

TASD is a distributed system so all the services need to access common data. In this project, MongoDB has used a NoSQL database technology. MongoDB database cluster has been set up on Mongo Atlas. It is a multi-cloud database service that simplifies deploying and managing databases while offering the versatility we need to build resilient and performant global applications on cloud providers .

NoSQL is a generic term used to describe any data store that does not use a legacy approach of related tables of data. NoSQL databases store data as the objects used in applications. So, the form of the data does not need to translate into the data taken in the code. MongoClient class of pymongo Python library has been used in the project script.

3.4 Experiments

The mitigation solution should be used to recognize the Yo-Yo attack from benign traffic. Before implementation of the actual solution, the behavior of a classical DoS attack and a Yo-Yo attack on a cloud service with auto-scaling feature will be tested. The first experiment simulates a DoS attack on a web service running on EC2 instances inside an auto-scaling group. Then, another test will be applied to show the effect of the Yo-Yo attack on the same service and resources. Finally, the same Yo-Yo attack scenario will be simulated to the web service shielded by the suggested TASD solution. The test will also evaluate the dynamic trust value feature. The project source code and implementation can be found in the project Github repository using the below URL:

https://github.com/meraj-kashi/aws-auto-scaling-flask-app

3.4.1 Experiment 1

The first experiment is implemented to show the effect of DoS attack on the auto-scaling mechanism. The Locust tool has been used to send burst traffic toward the cloud service. In the first experiment, it is configured with the peak number of concurrent 10 users and a spawn rate of two users per second. It means that when Locust starts sending burst traffic with this configuration, it will spawn two new users every second until it fulfills the total number of 10 users to simulate.

3.4.2 Experiment 2

The second experiment focuses on the simulation of a Yo-Yo attack on a cloud service with auto-scaling feature. The Locust tool has been used to send burst traffic toward the cloud service. It is configured with the peak number of concurrent 10 users and a spawn rate of two users per second. It means that when Locust starts sending burst traffic with this configuration, it will spawn two new users every second until it fulfills the total number of 10 users to simulate. According to [2], the attacker should approximate the auto-scaling state and configuration in order to maximize the damage. The attacker can send prob requests and compare response time to detect the scale-out status. The same technique is suggested to detect scale-in. The suggested solution has been implemented to detect scaling intervals in this experiment. A Python script has been developed to send probe requests and store response time.

3.4.3 Experiment 3

This is the actual experiment to test the Yo-Yo attack detection and mitigation system. The experiment focuses on whether the TASD system is able to differentiate malicious Yo-Yo traffic from benign traffic. The system is made to evaluate traffic requests' numbers and added a trust number to each client based on the algorithm 1 and 2. Figure 3.5 shows the experiment architecture diagram. In the algorithm 1, T_{init} is set to 10, and M is configured to 100 to update the trust value. Also in the algorithm 2 $T_{suspicious}$ and $T_{malicious}$ thresholds set to 7 and 5 respectively.

Using Terraform modules helps to deploy a new solution quickly. A new EC2 AMI running the TASD service has been created. TASD python code run as a systemd service in the background. Figure 3.6 illustrates the services inside the EC2 AMI and figure 3.7 shows the distributed TASD system overview.

The Locust tool has been used to send burst traffic toward the cloud service. It is configured with the peak number of concurrent 10 users and a spawn rate of two users per second. It means that when Locust starts sending burst traffic with this configuration, it will spawn two new users every

second until it fulfills the total number of 10 users to simulate. During the experiment, sending and stopping burst traffic to simulate the Yo-Yo attack have been done manually. To simulate the benign traffic, a Python script has been developed to send GET requests to the web service in a random interval between 1-5 seconds. The Python script runs in an instance running in ALTO cloud (Oslomet Openstack Cloud).

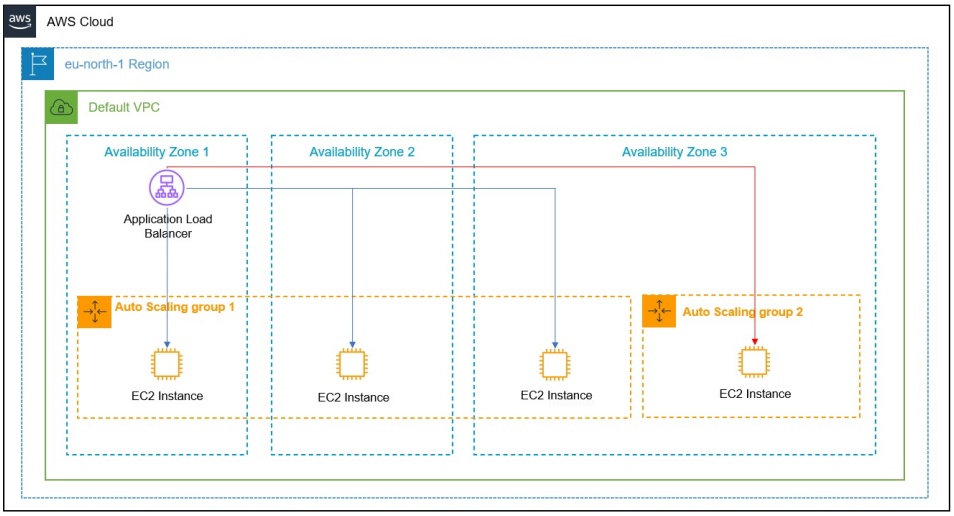

Figure 3.5: The project architecture diagram of the deployment TASD system on AWS cloud.

3.4.4 Experiment 4

This experiment is implemented to test the dynamic trust value feature. The original TASD system could add a trust value to each client based on the number of requests. According to section 3.2, the original TASD system introduced by [70] may have a false positive error. To optimize the performance of assigning the trust value, the algorithm 2 is modified to algorithm 3.

TASD system has been deployed using the AMI created in the previous experiment. The Locust tool has been used to send burst traffic toward the cloud service. It has been configured with the peak number of concurrent 10 users and a spawn rate of two users per second. It means that when Locust starts sending burst traffic with this configuration, it will spawn two new users every second until it fulfills the total number of 10 users to simulate. In the algorithm 1, T_{init} is set to 10, and M is configured to 100 to update the trust value. Also in the algorithm 3 $T_{suspicious}$ and $T_{malicious}$ thresholds set to 7 and 5 respectively. The waiting time to increase the trust value, $t_{release}$, is set to 15 minutes.

During the experiment, sending and stopping burst traffic to simulate

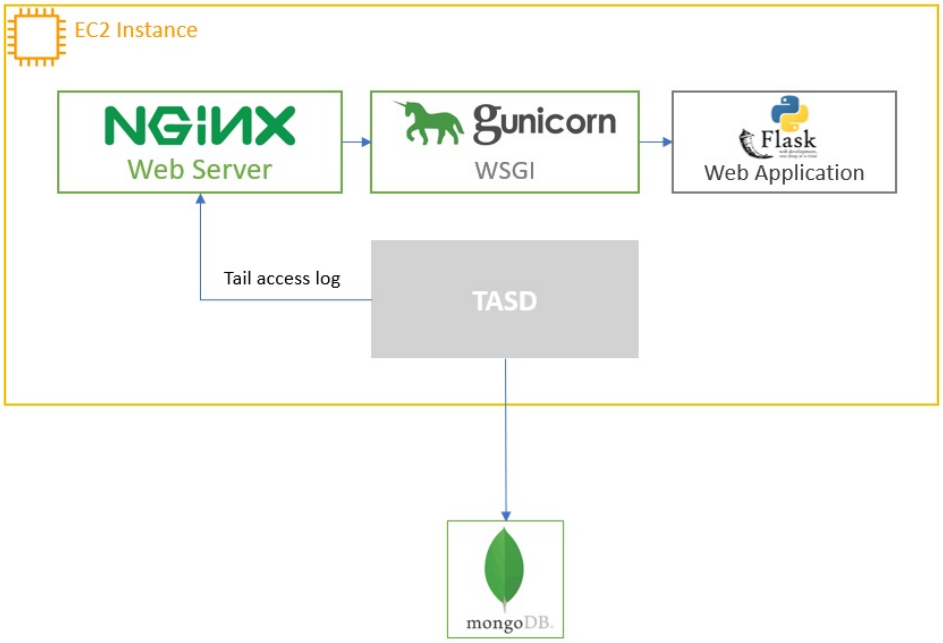

Figure 3.6: TASD services high level design running on EC2 instance including Nginx web server, Gunicorn WSGI, sample Flask web application and TASD as a side-car service.

the Yo-Yo attack have been done manually. To simulate the benign traffic, a Python script has been developed to send GET requests to the web service in a random interval between 1-5 seconds. The Python script runs in an instance running in ALTO cloud (Oslomet Openstack Cloud).

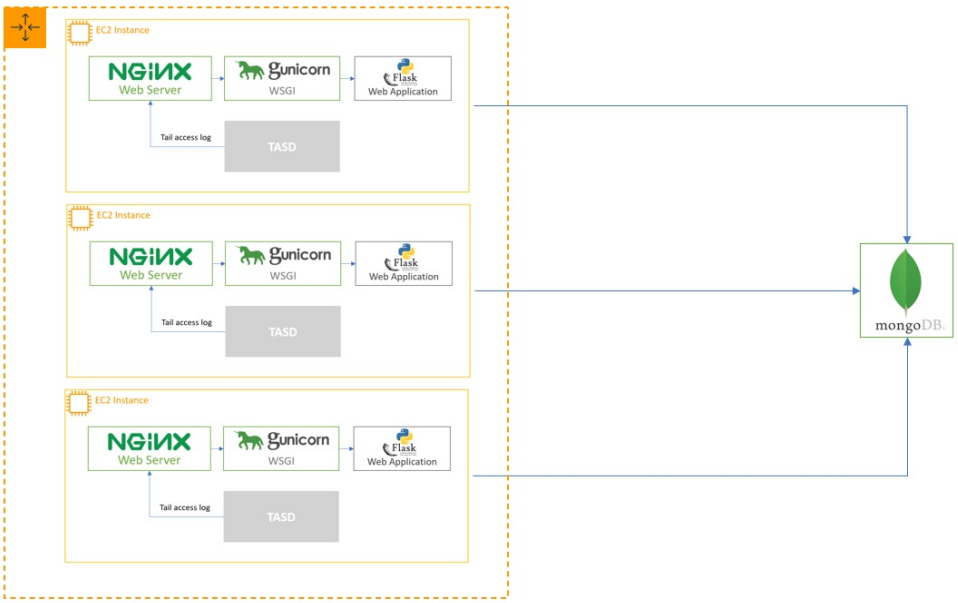

Figure 3.7: TASD services high level design running inside AWS auto-scaling group.

Chapter 4

Results

This chapter shows the result of the experiments. From the first experiment, we will see the effect of the DoS attack on a web service running in AWS Auto scaling. The second experiment result shows the difference between the DoS and Yo-Yo attacks. Finally, the experiment result of the TASD system will be proposed. The result will investigate, to a certain degree, if the proposed algorithms are able to deal with mitigating Yo-Yo attacks.

4.1 Experiment 1

4.1.1 DoS attack and auto-scaling

The first experiment was a DoS attack attempt on a web service running on an EC2 instance inside AWS Auto scaling group. Locust sent burst traffic to the AWS Application Loadbalancer DNS record. Figure 4.1 shows the total requests per second sent with Locust as a DoS attack traffic. The graph has been plated based on the data generated by Locust.

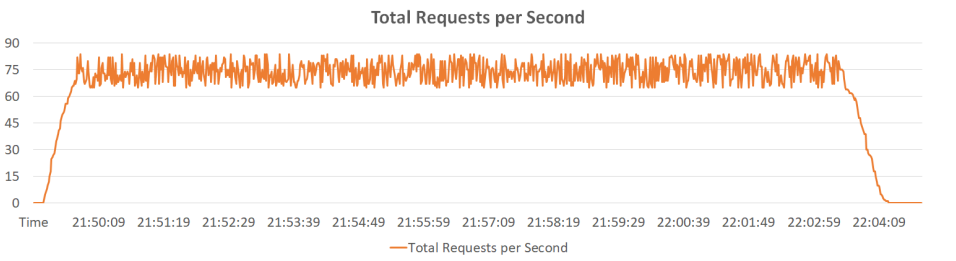

Figure 4.1: Total requests per second sent from Locust as a burst traffic.

AWS CloudWatch metric enabled to capture metrics from AWS Application Load balancer. RequestCount metric is the number of requests processed over IPv4 and IPv6. This metric is only incremented for requests where the load balancer node was able to choose a target. The statistics are

set to sum according to the AWS recommendation [24]. Figure 4.2 shows the sum of the requests received by each instance in a target group assigned to the load balancer. The red line shows the threshold set as a scale-out trigger for the auto-scaling policy.

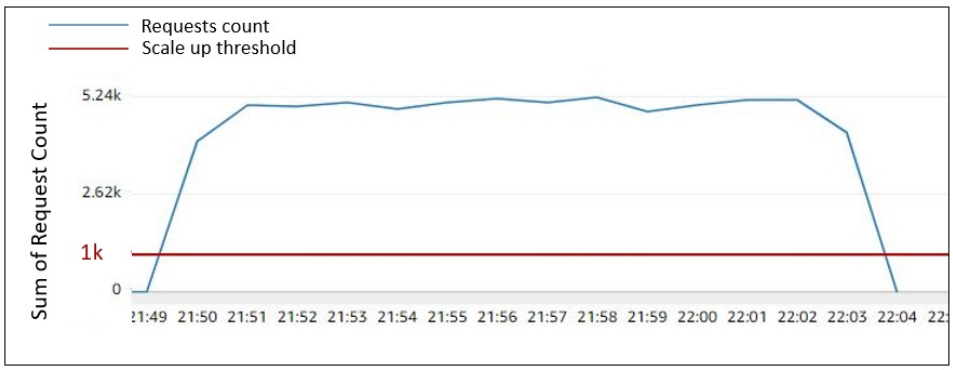

Figure 4.2: The sum of requests received by each instance in a target group assigned to the Load balancer.

Amazon EC2 publishes data points to CloudWatch that describe the auto-scaling instances. The metrics are available at one-minute intervals. In the project experiment, two metrics from auto-scaling have been observed. The GroupInServiceInstances metric is the number of instances that are running as part of the auto-scaling group. This metric does not include instances that are pending or terminating. So, to calculate the instance warming time, another metric called GroupDesiredCapacity was also observed. It shows the number of instances that the auto-scaling group attempts to maintain. The interval for Amazon EC2 instance monitoring is configurable. Figure 4.3 shows both metrics. The graph obviously shows the time takes from deploying an EC2 instance until the service is getting up.

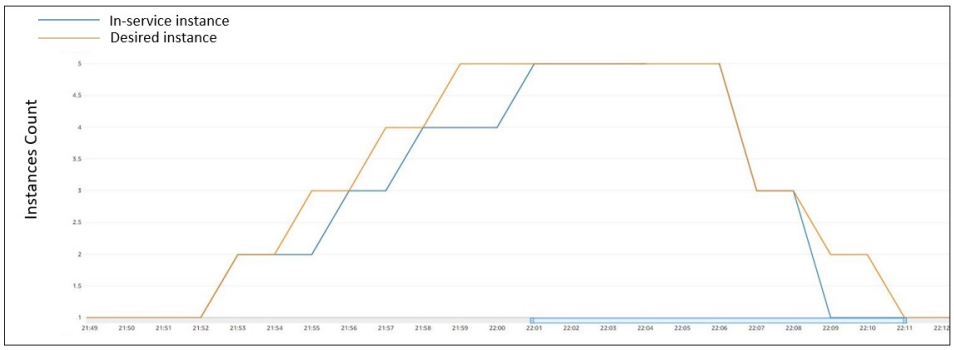

Figure 4.3: Auto-scaling group instance count including the in-service and desired capacity.

4.2 Experiment 2

4.2.1 Yo-Yo attack and auto-scaling

The second experiment was a Yo-Yo attack on a web service running on an EC2 instance inside AWS auto-scaling group. Locust sent burst traffic to the AWS Application Loadbalancer DNS record. Figure 4.1 shows the total requests per second from Locust as Yo-Yo attack traffic. The graph has been plated based on the data generated by Locust.

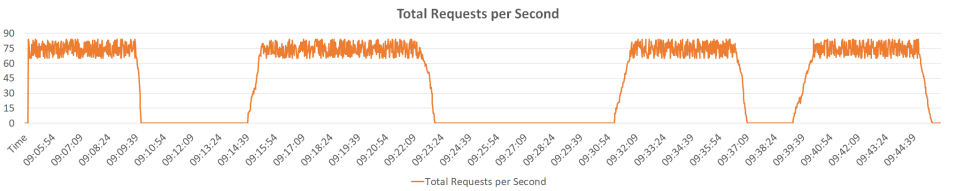

Figure 4.4: Total requests per second sent from Locust as a burst traffic.

Figure 4.5 shows the sum of the requests received by each instance in a target group assigned to the load balancer. The red line shows the threshold set as a trigger for the auto-scaling policy.

Figure 4.5: The sum number of requests received by each instance in a target group assigned to the Loadbalancer.

Figure 4.2.1 presents the number of in-service and desired EC2 instances in the auto-scaling group during the experiment.

4.2.2 Attacker probe response time

Based on the experiment description, the attacker sent a prob request to detect the scaling status. As the auto-scaling policy has been configured to scale based on the Request Count, then the response time of the probe requests was stable during the test. Figure 4.7 illustrates the response time from the probe request in milliseconds.

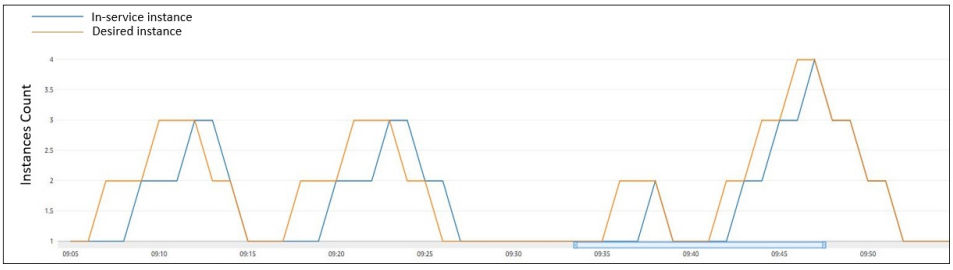

Figure 4.6: Auto-scaling group instance count including the in-service and desired capacity.

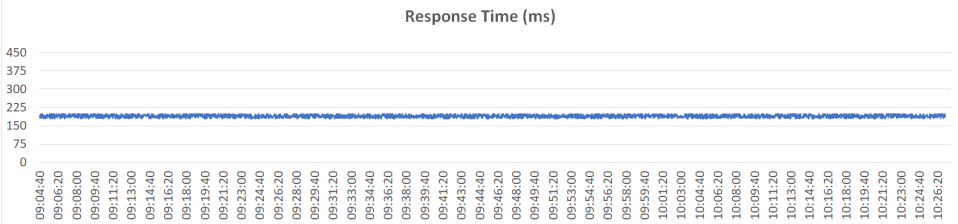

Figure 4.7: Response time from web service running on EC2 to the probe requests.

4.3 Experiment 3

4.3.1 TASD

In this experiment, a TASD system was run to detect and mitigate the Yo-Yo attack. In each simulation test, TASD considers each user to be benign and initializes the trust value of each user equal to 10, and the middle and minimum trust values are set to below seven and five respectively. Trust value was updated if the number of requests for the users in the high user set increased by 100 packets in scale-in. When the trust value of a user is lower than seven, TASD will consider the request from this user as suspicious and shift traffic to the second auto-scaling group. If the trust value of each user is lower than 5, TASD will consider the request from this user as malicious, and AWS VPC Access Control List (ACL) deny rule will be added for this specific user IP address.

Figure 4.8 illustrates the number of requests sent via Locust as an attacker toward the web application. The figure shows the AWS ALB traffic shifting and VPC ACL blocking IP address moments. Moreover, figure 4.9 shows the median response time from the web application captured by the attacker. The response time from instances running in auto-scaling group 1 is around 200 milliseconds. After decreasing the trust value to below 7, the traffic has been shifted toward the second auto-scaling group. As the web application was configured to respond with a two-second delay, the

response time doubled to 400 milliseconds. Finally, the trust value reached lower than 5, and the IP address was blocked in the AWS VPC ACL.

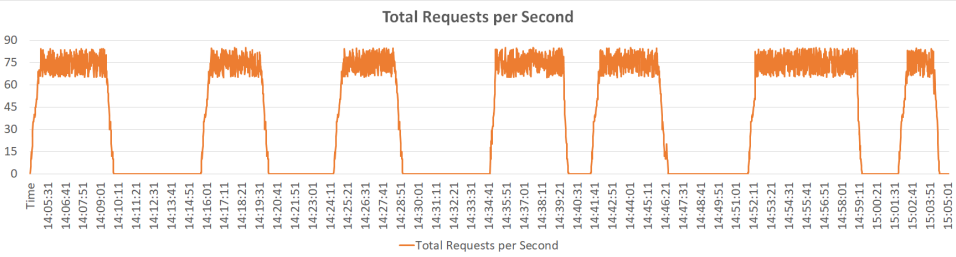

Figure 4.8: Total requests per second sent from Locust as a burst traffic.

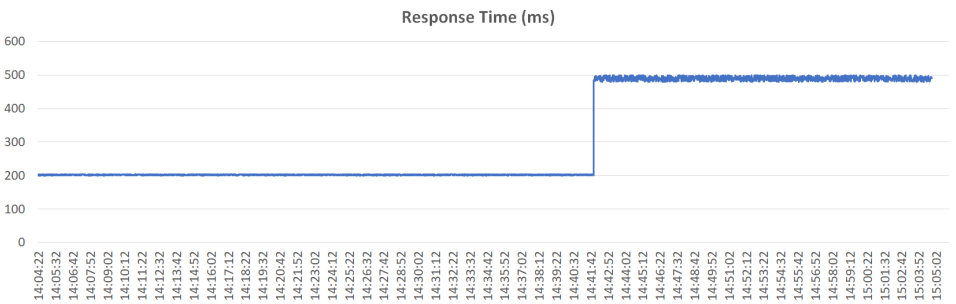

Figure 4.9: Web service response time from attacker side view.

Figure 4.10 presents the trust numbers assigned to the attacker during the experiment. It shows that the trust value of the attacker decreases as the number of updates performed by TASD increases. The trust value of the attacker dropped sharply and it can help the TASD system quickly mitigate the attack.

Figure 4.11 shows the sum of the requests received by each instance in a target group assigned to the load balancer. The red line shows the threshold set as a scale-out trigger for the auto-scaling policy.

TASD system decides to send traffic toward the auto-scaling group based on the trust value assigned to the clients. Figure 4.12 shows the number of in-service instance in auto-scaling group 1. The web application running inside this auto-scaling group is the main service for the clients. The figure shows that after shifting traffic toward the second auto-scaling group, the scaling is in a stable status. Figure 4.13 illustrates the number of instances inside the second auto-scaling group. Although the Yo-Yo attack causes scaling oscillation in the second auto-scaling, the main service running in the first auto-scaling is kept normal and stable.

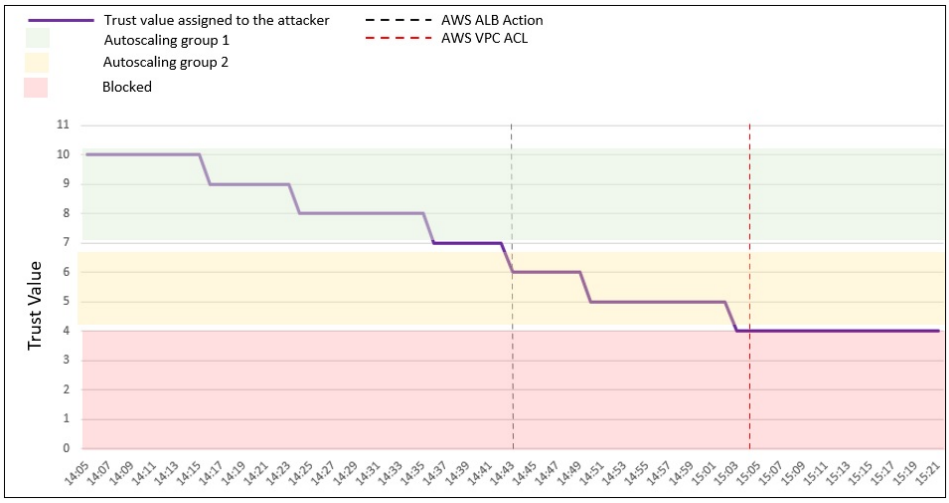

Figure 4.10: Trust value assigned to the attacker IP address during the experiment.

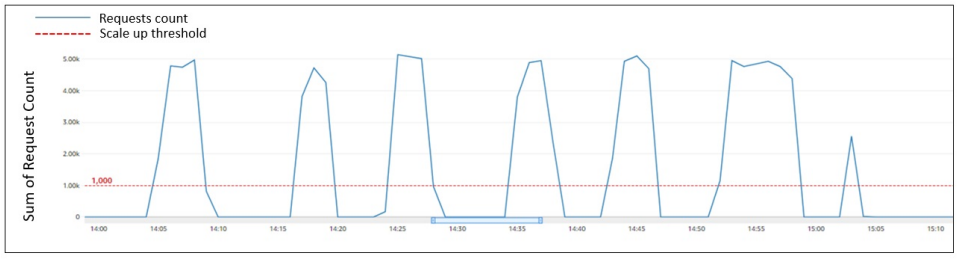

Figure 4.11: The sum of requests received by each instance in a target group assigned to the Loadbalancer.

4.4 Experiment 4

4.4.1 An optimization to TASD

As discussed in the section 3.2, the TASD solution may have a false positive error. If a benign user has malicious behavior for a while, TASD decreases trust value wrongly and it can cause service impact on the benign end user. The experiment has been done to evaluate the algorithm 3 for TASD optimization.

Figure 4.14 illustrates the number of requests sent via Locust as an attacker toward the web application. The figure shows the AWS ALB traffic shifting and VPC ACL blocking IP address moments. TASD calculates the trust value of each user and decides to send traffic to the correct autoscaling group or block the request. The blocking interval was set to 15 minutes. It means that the TASD increases the user trust value after 15 minutes of normal behavior. 4.15 shows the trust value of the attacker

Figure 4.12: Number of in-service instances in auto-scaling group 1.

Figure 4.13: Number of in-service instances in auto-scaling group 2.

during the experiment.

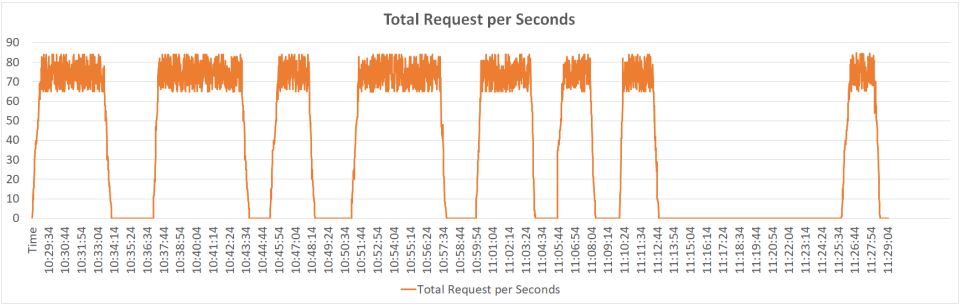

Figure 4.14: Total requests per second sent from Locust as a burst traffic.

4.16 illustrates response time of the attacker requests. It shows the response time from the instances inside the first auto-scaling group is approximately 200 milliseconds, and then the traffic shifted toward the second auto-scaling. As a result, the response time increased to 400 milliseconds. Then, the traffic is blocked by decreasing the trust value.

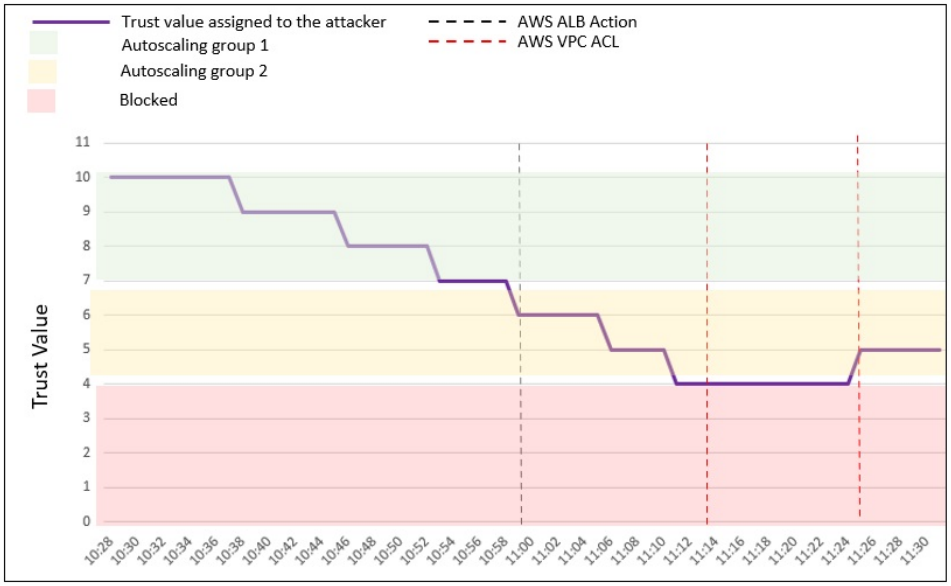

Figure 4.15: Trust value assigned to the attacker IP address during the experiment.

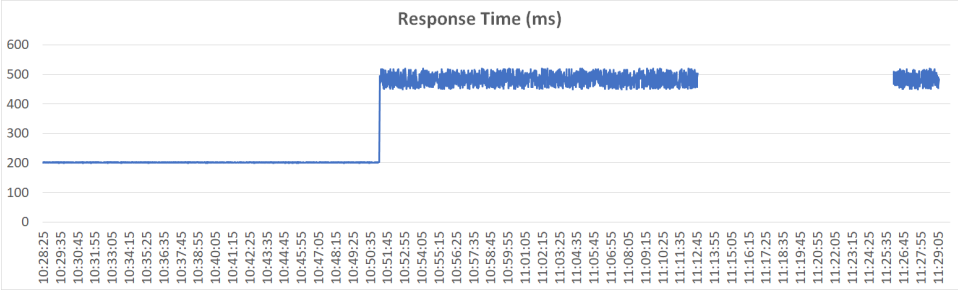

Figure 4.16: Web service response time from attacker side view. The gap in the graph shows the blocking interval.

Chapter 5

Discussion

This chapter discusses the advantages and disadvantages of the presented TASD solution to detect and mitigate the Yo-Yo attack and its results. Emphasis is put on the strengths and weaknesses of the system and how credible the results are. Also, the section presents future works and changes that should be applied to ensure an optimal solution.

5.1 DoS/DDoS attack and cloud auto-scaling

Cloud auto-scaling mechanism helps cloud services to meet the demands a workload makes for resources in order to maintain availability and performance as utilization increases. Sending bursts of traffic via DoS or DDoS attacks causes an increased load on the cloud services. If the auto-scaling policy has been configured to react properly based on the load increment, it can scale-out or scale-in the instances to support the amount of load. The auto-scaling policy allows the client to use scaling plans to configure a set of instructions for scaling their resources. During the project experiment, AWS Auto scaling policy was configured to use the Incoming Request Count metric. Also, the maximum instance number was set to five. In this case, as shown in figure 4.3, the instances increased to maximum capacity during the test. After reaching to maximum instance size, then the response time from the web application increased due to overwhelming the service. It means that the auto-scaling is not a solution to mitigate DoS/DDoS attacks.

5.2 Yo-Yo attack and cloud auto-scaling

The term Yo-Yo attack is due to oscillation from the on-attack to the off-attack phase. The experiment proved that the Yo-Yo attack can causes oscillation on instances inside the auto-scaling group. Figure 4.2.1 shows the instance oscillation during the experiment. The graph illustrates the time interval between the in-service instance and desired capacity. The interval was nominated as warming time. The warming time of scale-out is the time

that it takes to get ready to function and the warming time of scale-in is the time that an instance allocates to close all services and release resources. 4.2.1 shows that the warming time plays a major role in the damage, especially with the simple auto-scaling policy. Whenever the scaling metric threshold was selected close to the maximum capacity of the web application, the service degraded during scale-out because of warming time.

One approach to tackle the issue is to minimize the warming time. It is not capable for all scenarios, because some services take time to initialize. Another approach can be scale-out in two steps. It means that the auto-scaling increases the number of instances by two. A bottleneck for this option is to keep some unused capacity available for quick response. Under this approach, the user pays for the unused machines all the time. Another option is to scale-out early and scale-in slowly. In another word, there is no rush to scale-in the instance immediately.

The experiment also shows that using a probe request can not detect the scaling status necessarily. As the auto-scaling policy has been configured to scaling based on the Request Count, then the response time of the probe requests was stable during the test and it did not help the attacker to detect the scaling. As the auto-scaling policy has been configured to scale based on the Request Count, then the response time of the probe requests was stable during the test.

5.3 Yo-Yo attack and TASD solution

This project is an implementation of a Yo-Yo attack detection and mitigation system. The mitigation system consists of two modules to detect and mitigate the attack. Algorithm 1 is used in the detection module. The algorithm builds on the idea that the attacker sends burst traffic during scale-in and stops sending traffic in scaling-out. The concept of the trust value assigned to each user is a kind of user QoS (Quality of Services) implementation.

In a Yo-Yo attack, the attacker checks the response time from a probe request to check the cycle of scaling-out and scaling-in of the service. The experiment shows that if the metric threshold selects properly, then the response time is stable during scaling. Therefore, auto-scaling can break the cyclical nature of the attacker's probe request which prevents attackers from inferring the state of the auto-scaling mechanism.

TASD detection module works based on the difference between the user request number in scale-out and scale-in. The M value in the algorithm 1 is the difference value. It is important to select the proper value for M based on the client's cloud services workload. The value should be configured after analyzing the historical behavior of the users. The wrong value of M can cause extremely high false-positive errors.

TASD mitigation module can decrease the number of Yo-Yo attacks as well as the number of scaling. Figure 4.2.1 shows the instance scaling status for a non-TASD solution. It shows that during the Yo-Yo attack, instances cycle continuously. The same experiment on a system with the TASD solution shows that the scaling status is getting stable after detecting the attack and blocking the attacker. Figure 4.12 illustrates that after forwarding the traffic to the second auto-scaling group, as a defense mechanism, the number of instances stays at the stable status. According to the algorithm 2, forwarding to the second auto-scaling helps clients to continue their main workloads on the main servers. Moreover, the TASD can block the malicious server at the VPC level. Blocking at the VPC level helps to prevent increasing load on the load balancer. Therefore, TASD can efficiently mitigate the request load on the web application running on the cloud.

Algorithm 3 added an optimization to the initial TASD solution. To decrease the false-positive error of detecting a normal user as a suspicious or malicious user, TASD can update the trust value dynamically. The mitigation module can increase the trust value of a user after an interval.

5.4 Yo-Yo attack and Cost

One of the main discussions regarding the Yo-Yo attack and auto-scaling is the economic penalty. The Yo-Yo attack victim needs to pay for the extra resources required to process the bogus traffic and resources, which provide no real benefit to the victim. The cloud users also may affect the financial loss indirectly because of performance degradation and then causes the quality of services decrement.

5.5 Future Work

This section will address possible future work, which can make the approach presented in this thesis more robust when it comes to Yo-Yo attacks detection and mitigation. The implementation of the project was a Proof of Concept. According to section 3.2, the TASD detection module is a Python code that reads Nginx log files to capture request information. Cloudflare has published new open-source projects called LuaJIT and OpenResty [44]. Lua is the embeddable scripting language, which provides meta-mechanisms for implementing features, instead of providing a host of features directly in the language [4]. Cloudflare introduces a Just-In-Time compiler for the Lua language. LuaJIT has been used as a scripting middleware with a good performance in network applications [5]. OpenResty is a full-fledged web platform that integrates an enhanced version of the Nginx core and an enhanced version of LuaJIT. It is designed to help developers easily build scalable web applications, web services, and dynamic web gateways. OpenResty can be integrated with remote backend data-

base like MySQL, PostgreSQL, Memcached and Redis [6].

The database is one of the key components of the TASD project. As the TASD is a distributed system, it needs access to a common database. Using an in-memory database is a good option to continue with. For example, Redis can provide ingest and query time series of data. It supports a high volume of interactions with a low read latency [59]. Moreover, the Redis INCR command is used to increment the integer value of a key by one. These are some examples of how to improve the database performance in the TASD system.

Rather than optimizing the current approach, implementing a machine learning solution can improve the detection method. Machine learning may include two steps: feature extraction and model detection. In the feature extraction stage, the Yo-Yo attack traffic characteristics with a large proportion should be extracted by comparing the data packages classified and auto-scaling status. In the model detection stage, the extracted features are used as input features of machine learning, and a proper algorithm is needed to train the attack detection model.

Chapter 6

Conclusion

This master thesis project investigated on auto-scaling mechanism in cloud technology and how the auto-scaling is vulnerable to DoS/DDoS and Yo-Yo attacks. It is observed that sending burst traffic toward a service inside an auto-scaling group as a DoS attack causes scaling-out the instances. The scaling policy defines how to mitigate the load increment. Selecting the proper metrics, scale-out and scale-in interval and scaling strategy are important factors in the attack mitigation. In the other words, auto-scaling is a very powerful tool, but it can also be a double-edged sword. Without the proper scaling configuration and testing it can cost cloud users a lot. So, auto-scaling in cloud is not a remedy for DoS/DDoS attacks, and it is a trade of performance and cost.

Due to the mentioned characteristics of the auto-scaling, the Yo-Yo attack has appeared. The experiments showed that the Yo-Yo attack toward auto-scaling causes both performance and cost problems for cloud users. The oscillation between scale-out and scale-in status and having a high warm-up interval for instance can affect the service performance. On the other hand, The cloud user needs to pay the cloud infrastructure provider for the extra resources required to process the bogus traffic and resources, which provide no real benefit to the victim. It opens the question of how to detect and mitigate the Yo-Yo attack. This thesis was an implementation of the suggested Trust-based Adversarial Scanner Delaying (TASD) solution at [70] to detect and mitigate Yo-Yo attacks. The approach has been tested, and optimization has been applied to the initiate algorithms.

The TASD system assigns a trust value to each user and updates the trust value based on the clients' behavior. As a defense approach, based on the trust value of each user, it can forward traffic toward a delayed system or deny access to the user completely. The project has been implemented on the AWS cloud services and the results show that the TASD system can detect and mitigate the Yo-Yo attack. To decrease the false-positive error in the TASD system, we recommended adding a dynamic trust value that can increase or decrease based on the user behavior.

Bibliography

[1] *"Yo-Yo" DDoS Attacks: How to Defeat Them*. https://www.reblaze.com/blog/ddos-protection/yo-yo-ddos-attacks-how-to-defeat-them/. Accessed: 15th May 2022.

[2] A. Bremler-Barr, E. Brosh, and M. Sides. 'Ddos attack on cloud auto-scaling mechanisms'. In: (May 2017), pp. 1–9.

[3] *About App Autoscaler*. https://docs.pivotal.io/application-service/2-7/appsman-services/autoscaler/about-app-autoscaler.html. Accessed: 15th May 2022.

[4] *About Lua*. https://www.lua.org/about.html. Accessed: 15th May 2022.

[5] *About LuaJIT*. https://luajit.org/luajit.html. Accessed: 15th May 2022.

[6] *About OpenResty Project*. https://openresty.org/en/. Accessed: 15th May 2022.

[7] Alethea Toh. *Azure DDoS Protection—2021 Q3 and Q4 DDoS attack trends*. https://azure.microsoft.com/en-us/blog/azure-ddos-protection-2021-q3-and-q4-ddos-attack-trends//. Accessed: 15th May 2022. Jan. 2022.

[8] Alibaba Clouder. *DDoS Attack Statistics and Trend Report by Alibaba Cloud*. https://www.alibabacloud.com/blog/ddos-attack-statistics-and-trend-report-by-alibaba-cloud_597607. Accessed: 15th May 2022. Apr. 2021.

[9] Wael Ali Alosaimi. *A Security Framework for Preventing Denial of Service and Economic Denial of Sustainability Attacks in the Cloud Computing Environment*. eng. 2016.

[10] *Amazon EC2 Instance Types*. https://aws.amazon.com/ec2/instance-types/. Accessed: 15th May 2022.

[11] *Amazon EC2 On-Demand Pricing*. https://aws.amazon.com/ec2/pricing/on-demand/. Accessed: 15th May 2022.

[12] Amir Dahan. *Are You Protected Against Burst Attacks?* https://blog.radware.com/security/2018/02/burst-attack-protection/. Accessed: 15th May 2022. Feb. 2018.

[13] *An open source load testing tool*. https://locust.io/. Accessed: 15th May 2022.

[14] 'An Overview of the AWS Cloud Adoption Framework'. Version 2. In: (2017), p. 23.

[15] Ankur Jain, Asst. Prof. Dr. L. K. Vishwamitra. 'A Review on Cloud Data Security using Frequency Variation and Hard Logarithmic based Algorithm'. In: 6 (June 2017). ISSN: 2347-6435. URL: https://doi.org/10.1007/s00521-016-2317-5.

[16] *Annual Cybersecurity Report*. https://www.cisco.com/c/dam/m/hu_hu/campaigns/security-hub/pdf/acr-2018.pdf. Accessed: 15th May 2022.

[17] *Application layer DDoS attack*. https://www.cloudflare.com/learning/ddos/application-layer-ddos-attack/. Accessed: 15th May 2022.

[18] *Are You Protected Against Burst Attacks?* https://blog.radware.com/security/2018/02/burst-attack-protection/. Accessed: 15th May 2022.

[19] S. M. Bellovin. 'Security Problems in the TCP/IP Protocol Suite'. In: *SIGCOMM Comput. Commun. Rev.* 19.2 (Apr. 1989), pp. 32–48. ISSN: 0146-4833. DOI: 10.1145/378444.378449. URL: https://doi.org/10.1145/378444.378449.

[20] Mitko Bogdanoski, Tomislav Shuminoski and Aleksandar Risteski. 'Analysis of the SYN flood DoS attack'. In: *International Journal of Computer Network and Information Security* 5 (June 2013), pp. 1–11. DOI: 10.5815/ijcnis.2013.08.01.

[21] C.Hoff. *Cloud Computing Security: From DDoS (Distributed Denial Of Service) to EDoS (Economic Denial of Sustainability)*. https://rationalsecurity.typepad.com/blog/2008/11/cloud-computing-security-from-ddos-distributed-denial-of-service-to-edos-economic-denial-of-sustaina.html. Accessed: 15th May 2022.

[22] Victor Chang, Yen-Hung Kuo and Muthu Ramachandran. 'Cloud computing adoption framework: A security framework for business clouds'. In: *Future Generation Computer Systems* 57 (2016), pp. 24–41. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2015.09.031. URL: https://www.sciencedirect.com/science/article/pii/S0167739X15003118.

[23] *Cisco Security Appliances Targeted for DoS Attacks via Old Bug*. https://www.bleepingcomputer.com/news/security/cisco-security-appliances-targeted-for-dos-attacks-via-old-bug/. Accessed: 15th May 2022. Dec. 2019.

[24] *CloudWatch metrics for your Application Load Balancer*. https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-cloudwatch-metrics.html. Accessed: 15th May 2022.

[25] Viktor Danielsen. 'Detecting Yo-Yo DoS attack in a container-based environment'. Master's Thesis. Oslo Metropolitan University, 2021.

[26] Marwan Darwish, Abdelkader Ouda and Luiz Fernando Capretz. 'Cloud-based DDoS attacks and defenses'. In: *International Conference on Information Society (i-Society 2013)*. 2013, pp. 67–71.

[27] Ronen Ben David and Anat Bremler Barr. 'Kubernetes Autoscaling: YoYo Attack Vulnerability and Mitigation'. eng. In: (2021).

[28] *Denial-of-service attack*. https://en.wikipedia.org/wiki/Denial-of-service_attack. Accessed: 15th May 2022. Aug. 2021.

[29] Rashmi V. Deshmukh and Kailas K. Devadkar. 'Understanding DDoS Attack its Effect in Cloud Environment'. In: *Procedia Computer Science* 49 (2015). Proceedings of 4th International Conference on Advances in Computing, Communication and Control (ICAC3'15), pp. 202–210. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.04.245. URL: https://www.sciencedirect.com/science/article/pii/S1877050915007541.

[30] Christos Douligeris and Aikaterini Mitrokotsa. 'DDoS attacks and defense mechanisms: classification and state-of-the-art'. In: *Computer Networks* 44.5 (2004), pp. 643–666. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2003.10.003. URL: https://www.sciencedirect.com/science/article/pii/S1389128603004250.

[31] *Dynamic scaling for Amazon EC2 Auto Scaling*. https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scale-based-on-demand.html. Accessed: 15th May 2022.

[32] Eldad Chai. *Hit and Run DDoS attack*. https://www.imperva.com/blog/hit-and-run-ddos-attack/. Accessed: 15th May 2022. May 2013.

[33] Eurostat. *Cloud computing - statistics on the use by enterprises*. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Cloud_computing_-_statistics_on_the_use_by_enterprises. Accessed: 15th May 2022. Dec. 2021.

[34] *GCP - Autoscaling groups of instances*. https://cloud.google.com/compute/docs/autoscaler. Accessed: 15th May 2022.

[35] Yatheendraprakash Govindaraju, Hector A Duran-Limon and Efrén Mezura-Montes. 'A regression tree predictive model for virtual machine startup time in IaaS clouds'. eng. In: *Cluster computing* 24.2 (2020), pp. 1217–1233. ISSN: 1386-7857.

[36] M. Guirguis et al. 'Reduction of quality (RoQ) attacks on Internet end-systems'. In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* Vol. 2. 2005, 1362–1372 vol. 2. DOI: 10.1109/INFCOM.2005.1498361.

[37] Gupta, B. B, Badve, Omkar P. 'Taxonomy of DoS and DDoS attacks and desirable defense mechanism in a Cloud computing environment'. In: *Neural Computing and Applications* 28 (Dec. 2017), pp. 3655–3682. ISSN: 1433-3058. DOI: 10.1007\/s00521-016-2317-5. URL: https://doi.org/10.1007/s00521-016-2317-5.

[38] *Horizontal Pod Autoscaling*. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/. Accessed: 15th May 2022.

[39] *IBM Auto Scale*. https://cloud.ibm.com/docs/virtual-servers?topic=virtual-servers-about-auto-scale. Accessed: 15th May 2022.

[40] *Internet Control Message Protocol*. https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol/. Accessed: 15th May 2022.

[41] Georgios Kambourakis et al. 'A Fair Solution to DNS Amplification Attacks'. In: *Second International Workshop on Digital Forensics and Incident Analysis (WDFIA 2007)*. 2007, pp. 38–47. DOI: 10.1109/WDFIA.2007.4299371.

[42] David Karig and Ruby Lee. 'Remote Denial of Service Attacks and Countermeasures'. In: (Oct. 2002).

[43] Kaspersky. *DDoS attacks hit a record high in Q4 2021*. https://www.kaspersky.com/about/press-releases/2022_ddos-attacks-hit-a-record-high-in-q4-2021/. Accessed: 15th May 2022. Feb. 2022.

[44] *Keeping our open source promise*. https://blog.cloudflare.com/keeping-our-open-source-promise/. Accessed: 15th May 2022.

[45] Ko R, Lee SSG. *Cloud Computing Vulnerability Incidents: A Statistical Overview*. https://downloads.Cloudsecurityalliance.org/initiatives/cvwg/CSA_Whitepaper_Cloud_Computing_Vulnerability_Incidents.zip/. Accessed: 2014. Aug. 2013.

[46] Sanjeev Kumar. 'Smurf-based Distributed Denial of Service (DDoS) Attack Amplification in Internet'. In: *Second International Conference on Internet Monitoring and Protection (ICIMP 2007)*. 2007, pp. 25–25. DOI: 10.1109/ICIMP.2007.42.

[47] F. Lau et al. 'Distributed denial of service attacks'. In: *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0*. Vol. 3. 2000, 2275–2280 vol.3. DOI: 10.1109/ICSMC.2000.886455.

[48] Wei-zhou Lu and Shun-zheng Yu. 'An HTTP Flooding Detection Method Based on Browser Behavior'. In: *2006 International Conference on Computational Intelligence and Security*. Vol. 2. 2006, pp. 1151–1154. DOI: 10.1109/ICCIAS.2006.295444.

[49] Malachi Kenney. *Ping of Death*. https://insecure.org/sploits/ping-o-death.html. Accessed: 15th May 2022. Jan. 1997.

[50] *Manual scaling for Amazon EC2 Auto Scaling*. https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-manual-scaling.html. Accessed: 15th May 2022.

[51] Microsoft. *Microsoft Cloud Adoption Framework for Azure*. https://azure.microsoft.com/en-us/cloud-adoption-framework/#overview. Accessed: 15th May 2022.

[52] *Microsoft Azure Autoscaling*. https://docs.microsoft.com/en-us/azure/architecture/best-practices/auto-scaling. Accessed: 15th May 2022.

[53] David Moore et al. 'Inferring Internet Denial-of-Service Activity'. In: *ACM Trans. Comput. Syst.* 24.2 (May 2006), pp. 115–139. ISSN: 0734-2071. DOI: 10.1145/1132026.1132027. URL: https://doi.org/10.1145/1132026.1132027.

[54] *NTP amplification DDoS attack*. https://www.cloudflare.com/learning/ddos/ntp-amplification-ddos-attack/. Accessed: 15th May 2022.

[55] *Openstak Heat/AutoScalings*. https://wiki.openstack.org/wiki/Heat/AutoScaling. Accessed: 15th May 2022.

[56] Srinivas Padmanabhuni et al. 'Preventing Service Oriented Denial of Service (PreSODoS): A Proposed Approach'. In: *2006 IEEE International Conference on Web Services (ICWS'06)*. 2006, pp. 577–584. DOI: 10.1109/ICWS.2006.102.

[57] *Ping flood (ICMP flood)*. https://www.imperva.com/learn/ddos/ping-icmp-flood/. Accessed: 15th May 2022.

[58] *Predictive scaling for Amazon EC2 Auto Scaling*. https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-auto-scaling-predictive-scaling.html. Accessed: 15th May 2022.

[59] *Redis TimeSeries*. https://redis.io/docs/stack/timeseries/. Accessed: 15th May 2022.

[60] Sam Cook. *DDoS attack statistics and facts for 2018-2022*. https://www.comparitech.com/blog/information-security/ddos-statistics-facts/. Accessed: 15th May 2022. Feb. 2022.

[61] *Service Auto Scaling*. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-auto-scaling.html. Accessed: 15th May 2022.

[62] Mor Sides, Anat Bremler-Barr and Elisha Rosensweig. 'Yo-Yo Attack: vulnerability in auto-scaling mechanism'. In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 2015, pp. 103–104.

[63] Gaurav Somani, Manoj Gaur and Dheeraj Sanghi. 'DDoS/EDoS attack in cloud: affecting everyone out there'. eng. In: *ACM International Conference Proceeding Series*. Vol. 8-10-. SIN '15. ACM, 2015, pp. 169–176. ISBN: 1450334539.

[64] Sadhu Sreenivasarao. 'Application Layer DDOS Attack Detection and Defense Methods'. eng. In: *Proceedings of Emerging Trends and Technologies on Intelligent Systems*. Advances in Intelligent Systems and Computing. Singapore: Springer Singapore, 2021, pp. 1–12. ISBN: 9789811630965.

[65] *Swarm mode overview*. https://docs.docker.com/engine/swarm/. Accessed: 15th May 2022.

[66] *SYN flood*. https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/. Accessed: 15th May 2022.

[67] Tania Lorido-Botran, Jose Miguel-Alonso, Jose A. Lozano. 'A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments'. In: (Oct. 2014), pp. 559–592.

[68] *UDP flood attack*. https://www.cloudflare.com/learning/ddos/udp-flood-ddos-attack/. Accessed: 15th May 2022.

[69] Wikipedia. *Hit and Run DDoS*. https://en.wikipedia.org/wiki/Hit-and-run_DDoS#:~:text=Hit%2Dand%2Drun%20DDoS%20is,bringing%20down%20the%20host%20server. Accessed: 15th May 2022. Aug. 2014.

[70] Xiaoqiong Xu et al. 'Towards Yo-Yo attack mitigation in cloud auto-scaling mechanism'. eng. In: *Digital communications and networks* 6.3 (2020), pp. 369–376. ISSN: 2352-8648.

[71] Takeshi Yatagai, Takamasa Isohara and Iwao Sasase. 'Detection of HTTP-GET flood Attack Based on Analysis of Page Access Behavior'. In: *2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*. 2007, pp. 232–235. DOI: 10.1109/PACRIM.2007.4313218.