Dynamic honeypot deployment in the cloud

Ivan Beres

Technical Report

RHUL–ISG–2022–2

11 April 2022

Information Security Group
Royal Holloway University of London
Egham, Surrey, TW20 0EX
United Kingdom

2110981

Dynamic honeypot deployment in the cloud

August 2021

Submitted as part of the requirements for the award of the
MSc in Information Security
at Royal Holloway, University of London.

Contents

List of Abbreviations & Acronyms

ELB	Elastic Load Balancer
WS	Webserver
WTG	Webserver target group
HTG	Honeypot Target Group
PS	Public Subnet
PRS	Private Subnet
RT	Route table
IG	Internet Gateway
WLB	Webserver Load Balancer
AMI	Amazon Machine Image
EC2	Elastic Compute Cloud
VPC	Virtual Private Cloud
AWS	Amazon Web Services
IDS	Intrusion Detection System
VPN	Virtual Private Network
SSH	Secure Shell
HaaS	Honeypot as a Service
API	Application Programming Interface

Executive Summary

Honeypots are security defence tools, fake hosts designed to lure attackers away from real systems and capture malware threat analytics and attacker behaviour data for later analysis. This project sets out to research honeypots, their efficacy and the state of the art of honeypot development. Based on the research, a novel honeypot deployment concept is designed, implemented, tested and analysed leveraging cloud technologies.

Introduction

Honeypots are security defence tools. They are fake hosts designed to lure attackers away from real systems and capture malware threat analytics and attacker behaviour data for later analysis. The efficacy of a honeypot in attack mitigation and collecting attack behaviour analysis lies in its ability to obfuscate itself as a real system. Attackers are often successful in identifying honeypots because of the limitations inherent to fake systems. Honeypots are a vital part of the defence against attacks on computer networks. Their ability to lure attackers away from real targets makes them a crucial security tool. However, attackers are coming up with new ways of identifying and taking over honeypots. In the never-ending race against novel attacks, honeypots and how we use them must also be further developed.

This project solves some of the inherent limitations of honeypots by designing, building and evaluating a novel honeypot deployment concept leveraging cloud technologies. This new concept, a small, substantial contribution in the field, shifts the approach of deploying honeypots into the cloud. It is a new development in how honeypots are used and deployed in the cloud reducing the maintenance costs of honeypots in mitigating attacks by relying on resources that do not exist when the attack is started.

In section one of the project, the efficacy of common honeypots is researched, and gaps are identified in the literature to explore the state of the art of honeypot development and to pinpoint the issues with common honeypots, how attackers can identify them and the lack of research in leveraging the possibilities of the cloud in honeypot deployment. Section two breaks down the issues identified to honeypot believability, security, availability, automation and resource usage, setting the objectives to deploy honeypots in a resource-aware, timely and stealthy manner to resist identification by attackers by making honeypots indistinguishable from legitimate hosts. A novel, dynamic honeypot deployment concept is designed and implemented on a cloud platform in section three. Tests are set up, executed, and test results are captured in section four to prove the feasibility of the novel honeypot deployment design. Section five contains the analysis of the test results, and section 6 concludes the project. In section seven, further research opportunities of interest are discussed.

1. Literature review: honeypots

Introduction

The following section explores the different types and classes of honeypots, their shortcomings in self-obfuscation, common issues with honeypot deployment and maintenance and how attackers can detect them.

What are honeypots

Honeypots are services or entire servers set up to lure and expose a target for attackers to interact with it, wasting their time and resources. A honeypot is an innovative information security tool designed to capture information on attacker behaviour, attack tactics and malware. This way, honeypots can detect and mitigate attacks and collect threat intelligence to be analysed to strengthen defences. A honeypot may be any private or internet-facing computer, virtual machine, server, or endpoint connected to a network. When interacting with a honeypot, attackers are unaware that they are trying to hack into a fake system. A honeypot server uses vulnerable services, operating system versions and misconfigured software as a lure to entice attackers for interaction. A honeypot must disguise itself as a real system for as long as possible. A honeypot can only be effective as a diversion away from real hosts or services on the network if an attacker is unaware that they are being misled. When attackers realise they are wasting their time, the honeypot is revealed to them as a fake system. Attacker's move away from the honeypot and look for other targets, or they may try to attack and take over the honeypot and use it to attack other hosts. Honeypots underpin the research on new attack vectors and exploits, aiding in keeping information security defences ready for the next attack. Honeypots improve security by enabling threat analysis through observation and logging of attacker actions and providing a decoy for attackers. They also overcome some of the limitations of traditional intrusion detection systems (IDS).

The problem with traditional IDSs is that they generate false positives due to the high sensitivity needed to capture all malicious traffic. Installing and configuring an IDS is a challenge and often results in a flood of tickets overloading the security team. In order to detect an attack, an IDS must be aware of the type of behaviour it has to detect in attack signatures. An IDS is unlikely to detect attacks that are unknown to it. Because honeypots receive only malicious traffic, they are less likely to generate false positives and are capable of capturing novel and previously known attacks as well (Peter & Schiller, 2008).

However, while the traffic hitting honeypots is likely malicious, it does not mean that all malicious traffic is hitting the honeypot. Despite honeypots presenting a low hanging fruit for attackers, they may not be interested in interacting with them. The honeypot may be compromised for being too obvious or for being online too long. Attackers may ignore it depending on their ultimate goal in compromising a network. Thus, while installing honeypots enhances intrusion detection by capturing novel attacks and reducing noise, IDSs are still necessary to detect malicious traffic not hitting the honeypots.

Types of honeypots

Their purpose classifies honeypots into production and research honeypots. Their interaction level classifies them into low-interaction and high-interaction honeypots.

Production honeypots

Easy to use production honeypots are deployed into the enterprise networks owned by organisations to improve security. Production honeypots deployed to the network among other production systems provide a decoy for attackers to interact with instead of attacking real systems by exposing vulnerable services. Production honeypots are usually low-interaction and provide limited information about attackers and their tactics.

Research honeypots

Research honeypots are complex, high-interaction systems that are difficult to maintain. Government institutions, military and research organisations deploy them to collect information about attacker behaviour, strategies and tactics in vulnerability exploitation. Research honeypots do not improve the security of an organisation directly. They aid organisations to strengthen their defences by researching the threats they face.

Low-interaction honeypots

Low-interaction honeypots emulate operating systems and services without allowing in-depth interaction with the system (Kambow & Passi, 2014). These honeypots are easy to deploy and require little maintenance. Low-interaction honeypots provide intelligence based on captured connection attempts by exposing potentially vulnerable services such as telnet, FTP and SSH. Organisations deploy low-interaction honeypots to detect sources of malicious activity. Information about attack attempts is captured and sent to the IDS. Attackers' IP addresses are then blocked to avoid further connection attempts. Nothing is stopping the attacker from repeating the attack attempt.

Open-source developments such as Honeyd, deployed to a single host, can virtualise an entire network of hosts and services. Interaction with the emulated services is logged, and the trap is sprung. Another example is HoneyDB, providing real-time data of honeypot activity for the honeypots on their network. HoneyDB serves threat information through their API into which organisations can tap in. This way, distributed threat intelligence is available for a fee (Deception Logic Inc, n.d.). Low-interaction honeypots are deployed by organisations and hobbyists alike. Because low-interaction honeypots block outbound network traffic to stop them from being used to attack other hosts on the network, attackers can detect them. Their effectiveness in threat intelligence and their time-wasting potential are limited.

High-interaction honeypots

High-interaction honeypots are full-fledged systems with which attackers can interact. These systems are often virtualised and provide vulnerabilities in-depth with the operating system and additional software and services available for interaction as they would be on a real system. HoneySMB is a high-interaction honeypot for the Server Message Block protocol (R0hi7, n.d.). Lyrebird is a high-interaction, containerised honeypot framework exposing vulnerable applications (lyrebird, n.d.). High-interaction honeypots require high-level expertise, more computing resources and are more difficult to maintain when compared with low-interaction honeypots. While high-interaction honeypots are more difficult to detect, attackers may be able to take control over high-interaction honeypots and use them to attack other hosts.

Honeynets

Honeynets are decoy networks with one or more honeypots deployed. Honeynets are designed to gather intelligence and identify attackers and redirect them from attacking the enterprise network by providing vulnerable services on a decoy network. Honeynets allow outbound network traffic and traffic redirection between honeypots making it more difficult for an attacker to recognise that they are interacting with a fake network. Any host in the honeynet may be a point of ingress for attackers. The network with multiple, different honeypots has a better chance of catching more attackers due to exposing more vulnerable services when compared to a single honeypot.

Honeypot as a service

Recently, research has been conducted into delivering high-interaction honeypots as a service to reduce implementation and maintenance costs and increase attack mitigation effectiveness, recognising the difficulty in honeypot maintenance and obfuscation (Jafarian & Niakanlahiji, 2020). With HaaS, organisations outsource the generation, configuration and maintenance of honeypots. The generation of honeypots depends on the type and purpose of the enterprise network.

The HaaS provider will need access to the organisation's network to map the enterprise network to generate believable honeypots. This access may violate regulation and compliance depending on data protection legislation such as GDPR and other organisational policies. The outsourcing of hosting honeypots strips an organisation of the benefits of threat intelligence analytics. Data may be shared between the HaaS service provider and the client organisation. However, the ingesting, processing and analysis of that data would require additional resources, removing the benefits of outsourcing in the first place. For an organisation preferring to outsource its honeypot infrastructure, this information may be of little use without the know-how and the means to act. While the paper concludes that honeypots generated by the HaaS are less likely to be discovered to be fake hosts, it does not address HaaS power consumption.

Because in a HaaS, honeypots are hosted externally to the enterprise network, the IP addresses of network hosts have to be randomised at regular intervals to protect discovery by network mapping. The problem with this approach is that attackers are likely to discover that this is a HaaS just by detecting IP address randomisation. The HaaS solution does not mitigate attacks if the enterprise network is compromised.

Honeypot detection

The effectiveness of honeypots in attack mitigation lies in their ability to act as real systems and not be discovered as decoys. If the decoy environment does not match the attacker's mental model of how real enterprise networks should look like, the game is up (Tsikerdekis et al., 2019). Once discovered, honeypots become useless as defence tools. Understanding the methods and effort required to detect honeypots is vital for organisations, security professionals and honeypot developers.

Attackers often research their target, gathering information about the organisation's business, which paints a picture of the possible infrastructure that target mapping may reveal. Irrelevant, out-of-context services running on outdated, highly vulnerable hosts may be an easy giveaway for attackers. For example, if a security services provider operates an internet-facing Telnet service or an organisation without an e-commerce business hosts a webshop on a vulnerable web server, it is easy for attackers to recognise a honeypot.

System-level detection

System-level detection of honeypots requires access to the operating system and the privilege to execute arbitrary code. Attackers can identify honeypots by listing the software installed on a server and detecting virtual environments (S. Mukkamala, 2007). The uptime of a specific host may be significant for an attacker in determining the importance of a system. A server with a long uptime is likely critical to the organisation, while servers that are often restarted may be less important. This detail is overlooked in research concerning real-time self-configuring honeypot systems (Baykara & Das, 2018). Detecting sudden, arbitrary changes to the network may signal to an attacker that they have been discovered.

Network-level detection

An attacker observing the network can determine whether a host is legitimate based on its network activity. Low-interaction honeypots have no outbound network activity and may be identifiable on a network of hosts with regular network activity between them. Server uptime can be another giveaway. Important servers are rarely restarted, and attackers may ignore servers with a short uptime and move on to other targets.

Honeypots may be discovered by remote fingerprinting. Fingerprinting involves analysing network traffic or scanning the ports of a target system to collect as much information as possible. Honeyd is a commonly deployed, low-interaction honeypot capable of simulating a network of any size that can be fingerprinted by measuring the emulated network link latency (Fu et al., 2006). Emulating real-world systems is difficult. Honeypots should be deployed in a way to blend in with the rest of the network.

The detection of fingerprinting attacks in common honeypot deployments is vital in extending the lifespan of the system. Identified honeypots lose their purpose and can also be converted into zombies that attackers use to participate in other attacks against other systems. A fingerprinting attack against low-interaction honeypots may be detected using fuzzing while the attack is occurring. This technique may help in obfuscating the honeypot against fingerprinting. However, the detection method only works on known fingerprinting techniques and is ineffective against unknown methods (Naik et al., 2018).

Honeypots in the cloud

Deploying honeypots in the cloud is no different than deploying them on-premises or in regular data centres. Engineers may deploy the various honeypot offerings in the cloud just as they would on other platforms. However, cloud services offer various integrated services that can be leveraged to enhance honeypot lifespan and functionality while reducing hosting and maintenance-related costs.

Honeypots are vital in building a resilient defence against attacks on hosting infrastructure. While organisations are going through cloud transformation by moving parts of or all of their servers into the cloud, there is little research on cloud honeypot deployment or the efficiencies possibly gained by deploying honeypots in the cloud. Most of the literature is concerned with designing complex solutions for obfuscating honeypots from attackers to extend their usefulness. Cloud-based honeypot offerings simulate a specific service in isolation. No research is concerned with fixing the root cause of the problem with honeypots, which is that they are not real systems and can be discovered.

Gap analysis summary

In the previous section, shortcomings of common honeypots were identified concerning their believability, difficulties in obfuscating them, and maintenance and administration. As their believability is increasing, honeypots draw higher resources and incur higher computing and maintenance costs. Cloud technologies must be leveraged in honeypot deployment to achieve a high level of believability without the high costs in computing and maintenance.

2. Problem Statement

To overcome the limitations of honeypots identified in the previous sections, this project sets out to design, build and test a novel honeypot provisioning system in the cloud. The objective is to enhance the believability of honeypots and reduce the time and effort required to build and maintain high-interaction honeypots by leveraging cloud services. These objectives will be achieved by analysing and meeting the system requirements outlined in the subsections below.

The cloud comes with the advantages of quick computing provisioning speeds and reconfigurable network architecture not possible to achieve on-premise. Quick instantiation speeds make it possible to deploy honeypots dynamically, only when required. The reconfigurable network architecture enables network segmentation and enhances security.

Resources taken up by honeypot design, building and maintenance are greatly reduced by leveraging the cloud. Honeypot management and orchestration are simplified in the cloud when compared to managing on-premise systems. While any honeypot deployed on-premise can be deployed into the cloud, dynamic provisioning of resources makes it possible to mirror existing production environments and use them as sandbox honeypots. The sandbox honeypot concept allows the capture of the effects of novel malware on real systems. It also provides analytics on how attackers interact with real hosts without putting the rest of the infrastructure to risk.

Believability

Provisioned honeypots shall be carbon copies of their real counterparts, virtually indistinguishable from real systems in design, resources, capabilities and behaviour. Attackers should not be able to tell if they are interacting with a honeypot. There should be no technical or functional difference between the genuine server and its honeypot copy. Provisioned honeypots shall be high-interaction production honeypots, positioned next to production systems, offering deep interaction for attackers. Services or operating systems shall not be simulated.

Security

Honeypots shall never become a liability or pose a risk to the rest of the network and other servers. Provisioned honeypots shall be isolated from real servers using network segmentation. If attackers gain access and control of the honeypot, it shall not be possible to use it to attack other servers on the network.

Availability

Honeypots shall be provisioned only when needed. Provisioning honeypots shall be reasonably fast for the system to react to an attack promptly. The startup speed of honeypot hosts is critical in their usefulness. If the honeypot is too slow to start, it will not be included in an attacker's target enumeration attempts or may be discovered as a fake host.

Automation

The provisioning and de-provisioning of honeypots shall be automatic to avoid the additional workload of maintaining more servers than necessary. The architecture shall allow programmatic reconfiguration of the honeypot servers and network components on demand. Creating honeypot servers as copies, including all hardware specifications and software components of real servers, shall be automated.

Energy and cost-saving

Running unused servers negates the possible resource efficiency gains of cloud hosting. Honeypots servers should only be up and running when needed. The system shall terminate honeypot servers when they are no longer needed, thus reducing CO2 emissions and costs.

Summary

By leveraging the speed, automation possibilities, availability and security advantages of the cloud, honeypot deployment and management is made dynamic, simpler, more secure and automatic. Common honeypot functionality is augmented by creating sandbox honeypots yielding more precise analytics of real-system interactions.

3. Design specification

Introduction

Cloud computing enables the on-demand deployment of information technology resources over the internet. The Amazon Web Services (AWS) global infrastructure is introduced in the following subsections, followed by the description and specification of each AWS cloud service included in the design.

AWS global infrastructure

The AWS architecture spans multiple physical locations called regions. Logical clusters of data centres make up regions with redundant power, networking, and connectivity called Availability Zones. There are multiple physically isolated Availability Zones in each AWS region. Availability Zones are interconnected with low latency, encrypted, redundant, highly available networking over dedicated fibre (Amazon Web Services, n.d.-e).

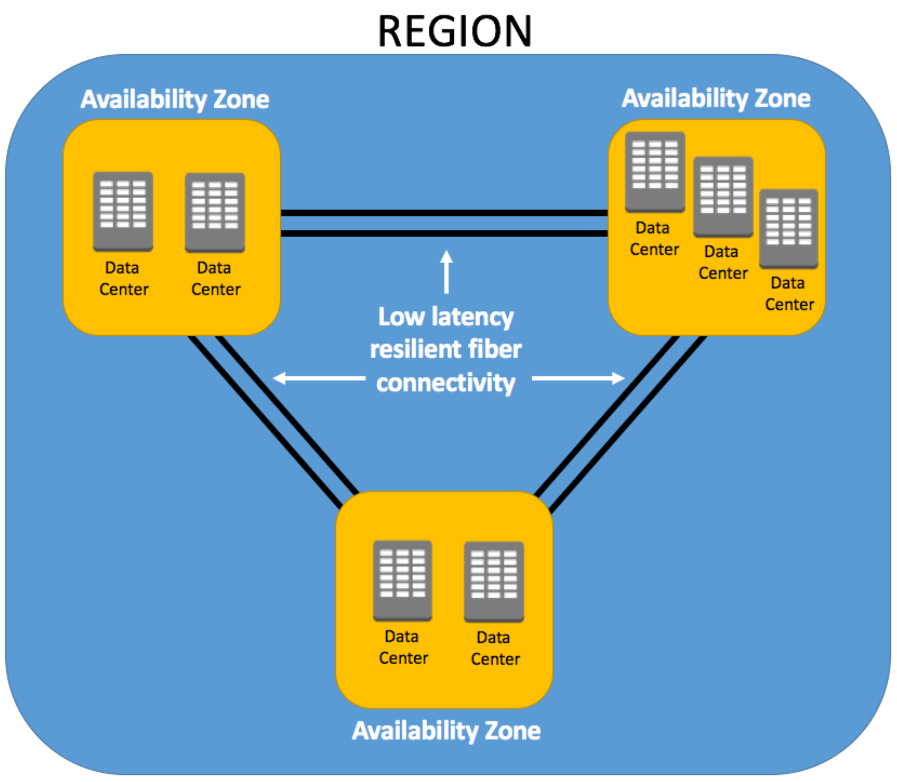


Figure 1 AWS Regions and Availability Zones (Scott, n.d.)

AWS Test environment

The test environment is comprised of virtual network components and virtual machines, including an internet-facing web server, a load balancer, an application server and a database server. The webserver is in a public subnet and is accessible from the internet through an internet gateway. The database and application servers are in a private subnet and are only accessible on specific ports within the Enterprise Network. While the test environment is a standard architecture, hosting a web application on AWS may be also be achieved by leveraging other AWS services.

Network components

Virtual Private Cloud (VPC)

AWS's VPC is a logically isolated virtual network sandbox where AWS resources can be placed (Amazon Web Services, n.d.-b). The systems architect can define IP ranges, subnets, network gateways and routing tables to design and build a custom virtual network. A VPC spans all availability zones in an AWS region and can have multiple subnets.

VPC specification

The Enterprise Network VPC is set up with an IPV4 CIDR block of 10.0.0.0/16 and includes two public subnets, PS1, PS2 and a private subnet PRS.

Subnets

Subnets in AWS reside in a single availability zone. Subnets are set up to provide security by network segmentation. Internet-facing web servers and honeypots are placed in public subnets, while database and application servers are placed into private subnets not reachable from outside the Enterprise Network. Each subnet is associated with a routing table with inbound and outbound traffic rules (Amazon Web Services, n.d.-i).

Subnet specification

Public subnet PS1 is associated with Web server WS1 and has an IPV4 CIDR block of 10.0.1.0/24. Public subnet PS2 is associated with Web server WS2 and has an IPV4 CIDR block of 10.0.3.0/24. Private subnet PRS is associated with the Database server DB1 and Application server AS1 and has an IPV4 CIDR block of 10.0.2.0/24.

Internet gateway

Internet gateways are redundant VPC components that are redundant and can scale horizontally. They allow network traffic between the internet and the VPC (Amazon Web Services, n.d.-f).

Internet gateway specification

The Internet gateway IG performs network address translation (NAT) for webservers WS1 and WS2 and provides a destination for the Enterprise Network VPC routing table for internet traffic.

Route tables

Route tables control how the VPC router directs network traffic. Each subnet must be associated with a routing table, but a routing table can be associated with multiple subnets.

Route table specification

Table 1 Routing table RT1 directs network traffic between PS1, PS2 and IG

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	IG

Table 2 Routing table RT2 directs network traffic between PRS and the local network

Destination	Target
10.0.0.0/16	local

Elastic Load Balancer

Because the project is focused primarily on defence against external scanning, the main concern in the test environment is the internet-facing web servers. Placing a load balancer in front of multiple web servers makes it possible to distribute traffic between them and add high availability, automatic scaling and fault tolerance to the web application (Amazon Web Services, n.d.-d). Furthermore, the listener of the Application Load Balancer type Elastic Load Balancer (ELB) on AWS enables the set up of forwarding rules based on IP addresses.

Elastic Load Balancer specification

The WLB ELB uses AWS Global Accelerator to expose the public IP addresses 52.223.23.164 and 35.71.153.62 to access the internet web application.

Listener rules enable the forwarding of requests coming to the ELB WLB to target groups. Listener rules can be created, updated and deleted programmatically using the Boto3 API. The Elastic Load Balancer can have up to 100 listener rules with five values each, limiting the list of IP address ranges to 500. Attacks conducted with over 500 IP address ranges at the same time would result in failed listener rule configuration updates. Unless multiple ELBs are used, such an attack could not be mitigated.

Target groups

Target groups add an extra level of modularity in network routing configuration, forwarding requests from the Elastic Load Balancer to their registered targets, such as EC2 instances. Targets in target groups are monitored via health checks by the ELB.

Target group specification

Target WTG and HTG enable the routing of requests to the EC2 web servers WS1 and WS2. Targets must be registered in a target group for the ELB to route requests to them. Target groups can be programmatically created and updated using the Boto3 API.

EC2 instances

Elastic Compute Cloud offers scalable, on-demand virtual machines in various capacities. EC2 is the main computing backbone of AWS, where all virtual servers reside (Amazon Web Services, n.d.-a). EC2 instances come in different hardware resources and sizes depending on their load and purpose.

EC2 instances specification

Web servers WS1 and WS2, database server DB1 and application server AS1 are t2.micro instances or virtual servers in the cloud to comply with the AWS free tier requirements. They are running the Amazon Linux 2 operating system with 1vCPU, 8GB of storage and 2GB of RAM. EC2 instances can be created from scratch, started, stopped or terminated at any time.

Webserver WS1 ID: i-0c9623abffe6688aa is running httpd server version: Apache/2.4.48 patched to the latest version. Webserver WS2 ID: i-0246b23f775ebc81b is running httpd server version: Apache/2.4.33 introducing some vulnerabilities (MITRE, n.d.). Database server DB1 is running MySQL, Application server AS1 is running PHP, and while they are part of the infrastructure, they are not in scope for this project.

Security Groups

Security groups control network traffic similarly to a stateful virtual firewall at the EC2 instance level. Inbound and outbound traffic rules can be set to allow network traffic on specific ports. Because security groups are stateful, responses to allowed outbound requests and requests to allowed inbound responses are allowed without specific inbound and outbound rules (Amazon Web Services, n.d.-h).

Security groups specification

Webserver Security group WSG sets inbounds and outbound rules for WS1 and WS2 EC instances. HTTP and HTTPS are allowed inbound to serve a website, and SSH is allowed for administration via a terminal client. All types of outbound traffic are allowed.

Table 3 Security group WSG inbound rules

Security group rule ID	IP version	Type	Protocol	Port range	Source
sgr-0f8c7f20cceb6cefa	IPv6	HTTPS	TCP	443	::/0
sgr-08d0cc3ca532250f0	IPv6	HTTP	TCP	80	::/0
sgr-0121fb3be11ab09ae	IPv4	HTTP	TCP	80	0.0.0.0/0
sgr-050d636ce6a5b6e1f	IPv4	HTTPS	TCP	443	0.0.0.0/0
sgr-0ab458a2e3821aacd	IPv4	SSH	TCP	22	0.0.0.0/0

Table 4 Security group WSG outbound rules

Security group rule ID	IP version	Type	Protocol	Port range	Source
sgr-01fc06a54be36dd24	IPv4	All traffic	All	All	0.0.0.0/0

Patching cycle

Patching security vulnerabilities during recurring monthly downtime is common practice in systems administration. Downtime scheduled to the same day and time window of each month is easily communicated to users. Unplanned downtime can occur when a serious vulnerability is discovered, and patching it is time-critical.

Creating and maintaining believable honeypots is time-consuming. Capturing the image and the vulnerabilities of servers before patches are applied is a quick and easy way of building a roster of real, in-context honeypot servers with real vulnerabilities that can be used at any time. Taking advantage of AWS's Amazon Machine Image (AMI) mechanism makes this possible. An AMI is a template containing the operating system, software applications and configuration of an EC2 instance. For example, an AMI can contain everything needed to run a webserver, including Apache, static content, and other configurations. Once started as an instance of the AMI, the new server will be ready to accept requests.

Before patching, new images are created for each server for future honeypot provisioning. While this process can also be done via automation, patching and creating new AMIs can be done manually as this does not occur often. Provisioning honeypot servers from AMIs greatly reduces the effort and time required in building and maintaining honeypot servers.

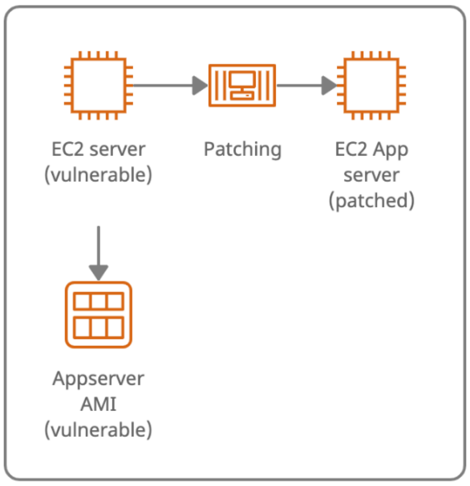


Figure 2 Patching cycle

Utilising AMIs to mirror the network partially or completely could enable pseudo-red-team security testing to allow dynamic patching resulting from frequent security assessments. The red team could dive deep into specific vulnerabilities identified through analytics captured from attacker interactions with sandbox honeypots and fully understand risk levels.

Lambda

Lambda is a serverless AWS compute service making it possible to run code without server provisioning, maintenance and administration. Lambda functions support, among many others, Python makes it easy to automate the repeated provisioning, configuration and maintenance of AWS services.

The Boto3 AWS SDK for Python provides an API and low-level access to provision and manage AWS services in Lambda. Lambda functions can be executed manually or can be set to execute when certain events occur. CloudWatch logs events can trigger Lambda functions.

Boto3

Boto3 is the AWS SDK for Python, consisting of the Botocore library for low-level access to AWS services and the Boto3 package to implement the Python SDK. Boto3 makes it possible to programmatically create, configure, and manage various AWS services (Amazon Web Services, n.d.-g).

Lambda functions

The Lambda functions below are based on the samples provided by AWS and are available in the Boto3 documentation (Amazon Web Services, n.d.-g).

Start_EC2 and stop_EC2: Defence mode step 2

The start_EC2 Lambda function starts a stopped EC2 instance when triggered. Existing honeypot EC2 instances can be started this way. The stop_EC2 Lambda function stops currently running EC2 instances.

Table 5 Start_EC2 Lambda function source code and description

`import boto3`	Importing boto3 SDK
`region = 'us-east-2'`	Setting AWS region
`instances = ['i-0f6521ce5f6dedea3']`	Variable to store array of instances to start
`ec2 = boto3.client('ec2', region_name=region)`	Create low-level service client
`def lambda_handler(event, context):`	Handler method to process start_instances event
`//Starting EC2 instance`	
`    ec2.start_instances(InstanceIds=instances)`	Event for starting instances
`    print('started your instances: ' + str(instances))`	Print function for logging
`//Stopping EC2 instance`	
`ec2.stop_instances(InstanceIds=instances)`	Event for stopping instances
`print('stopped your instances: ' + str(instances))`	

Start_AMI: Defence mode step 2

The start_AMI Lambda function creates a new honeypot EC2 instance from an existing AMI and starts it. Specific commands are executed upon startup to update the honeypot's installed packages captured in the AMI to match patch levels of real systems. Apache is installed and started so that the honeypot can fulfil its duties as a webserver, and uptime is obfuscated to avoid detection by attackers. The script specifies the subnet the honeypot instance will be part of and the virtual machine type.

Table 6 Start_AMI Lambda function source code and description

`import os`	`Import os Python module`
`import boto3`	`Importing boto3 SDK`
`region = 'us-east-2'`	`Setting AWS region`
`ec2 = boto3.client('ec2', region_name=region)`	`Create low-level service client`
`def lambda_handler(event, context):`	`Handler method to process start_instances event`
`    init_script = """#!/bin/bash`	`Variable to store commands executed after boot`
`                yum update -y`	`update installed packages to match patch versions`
`                yum install -y httpd24`	`install Apache httpd`
`               sudo systemctl start httpd`	`start httpd service`
`               sudo systemctl enable httpd`	`make sure httpd will start automatically`
`               sudo echo 0>/proc/sys/net/ipv4/tcp_timestamps`	`uptime obfuscation`
`    instance = ec2.run_instances(`	`Event to initialise and start EC2 instance`
`        ImageId= 'ami-0486cb8f557d6bdd5',`	`AMI image ID`
`        InstanceType='t2.micro',`	`Type of instance`
`        KeyName= 'ec2_2021_06_27',`	`Key for SSH access`
`        SubnetId= 'subnet-0037a16b193e6b2a6',`	`Subnet to associate the instance with`
`        InstanceInitiatedShutdownBehavior='terminate',`	`Upon shutdown, the instance is terminated`
`        UserData=init_script`	`Commands to be executed after boot`
`    )`	

HTG_update: Defence mode step 3

The HTG_update lambda script registers the newly provisioned honeypot instance into the HTG target group. HTG is the target for WLB's listener rule, forwarding requests from the IP addresses specified in the rule.

Table 7 HTG_update Lambda function source code and description

`import boto3`	`Importing boto3 SDK`
`client = boto3.client('elbv2')`	`Create low-level service client`
`def lambda_handler(event, context):`	`Handler method to process register_targets event`
`    body = {`	
`        "message": "Registering a new instance to target group HTG",`	
`        "input": event`	
`    }`	
`    response = client.register_targets(`	`Event to register target to HTG target group by specifying target HTG arn`
`    TargetGroupArn= 'arn:aws:elasticloadbalancing:us-east-2:542457226429:targetgroup/HoneypotTG/4814532c401c539f',`	
`    Targets=[`	
`        {`	
`            'Id': 'i-07d1075ae944609a8',`	`EC2 instance ID to register into target group`
`        },`	
`    ],`	
`)`	

ELB_update: Defence mode step 4

The ELB_update Lambda function creates a new listener rule for the ELB WLB. The new listener rule specifies the IP address as the source and the target group HTG as a destination to forward requests. This way, IP addresses deemed malicious by the IDS are forwarded to the honeypot EC2 instance WS2, part of the HTG target group. After the script is executed, the attacker interacts with a honeypot instead of a real system without realising it.

Table 8 ELB_update Lambda function source code and description

`import boto3`	Importing boto3 SDK
`client = boto3.client('elbv2')`	Create low-level service client
`def lambda_handler(event, context):`	Handler method to process create_rule event
`    body = {`	
`        "message": "Adding suspicious IP address to quarantine list",`	
`        "input": event`	
`    }`	
`    response = client.create_rule(`	Create listener rule event by specifying listener arn
`    ListenerArn='arn:aws:elasticloadbalancing:us-east-2:542457226429:listener/app/WebserverLB/5a2b7a5bf786ca1e/a82ef0acaacf173b',`	
`    Conditions=[`	
`        {`	
`        'Field': 'source-ip',`	Setting field type for the listener rule to source-ip
`        'SourceIpConfig':{`	
`        'Values': ['87.80.156.133/32',]`	Specifying the array of IPs to be included
`                }`	
`        }`	
`    ],`	
`    Priority=10,`	Rule priority
`    Actions=[`	
`        {`	
`            'TargetGroupArn': 'arn:aws:elasticloadbalancing:us-east-2:542457226429:targetgroup/HoneypotTG/4814532c401c539f',`	Target group to forward requests to
`            'Type': 'forward',`	Type of rule: forwarding in this case
`        },`	
`    ],`	
`)`	

CloudWatch

The CloudWatch service provides insights and logging and is used for monitoring most aspects of the AWS environment. In this project, CloudWatch is used to monitor and log events in the Firewall. These events are set to trigger Lambda functions to provision honeypots and update the listener rule configuration of the existing Elastic Load Balancer WLB.

Dynamic honeypot provisioning

The project assumes that network anomalies such as external port scanning attempts are detected successfully at the firewall level by a hypothetical IDS. Feedback and incident reports from such systems are used as part of the dynamic honeypot provisioning subsystem. While intrusion detection is an interesting and relevant topic, and existing systems may detect not all potential forms of intrusion, it is beyond the project's scope to explore these topics. This project focuses on events happening just after a network anomaly was detected.

Honeypots are provisioned when a network anomaly is detected. A Lambda function is triggered by a CloudWatch alert raised by CloudWatch ingesting logs coming from the Firewall. New virtual web, application and database servers have been provisioned that copy the existing servers based on Amazon Machine Images (AMI).

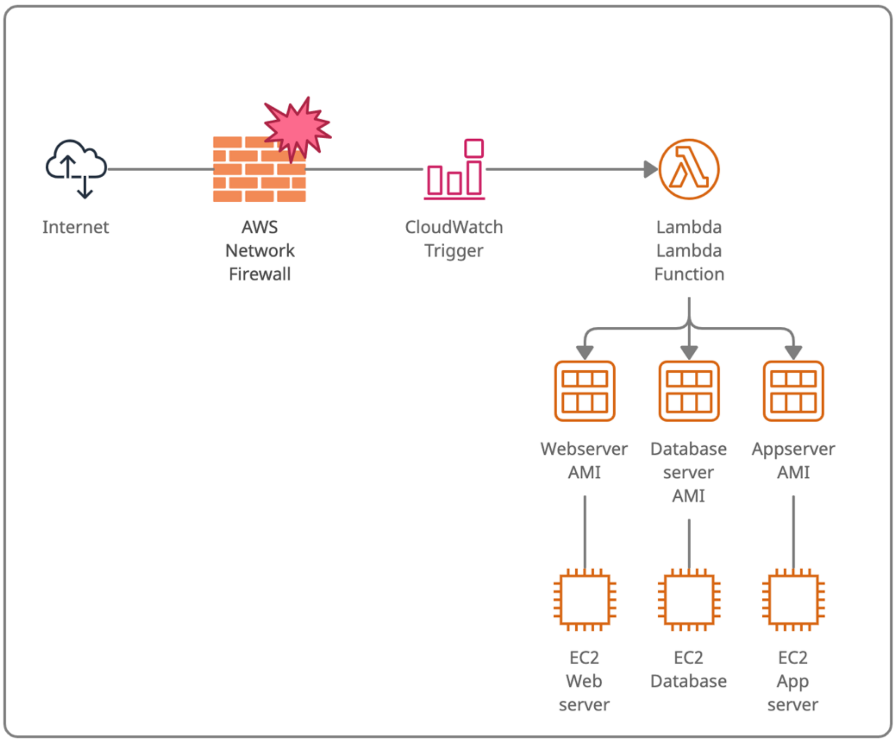


Figure 3 Honeypot provisioning

When honeypot provisioning is complete, each real server has a honeypot copy, potentially exposing a slightly more vulnerable software component or operating system version based on previous versions of the servers.

Elastic Load Balancer reconfiguration

When external network scanning is detected at the firewall level, CloudWatch raises a CloudWatch Alarm, triggering a Lambda function. The Lambda function takes the offending IP addresses from which the network scan was initiated and creates the Elastic Load Balancer WLB's listener rule to redirect all requests from the offending IP to Web server WS2, a newly provisioned honeypot. The listener rule is also updated to ensure that requests from non-offending IP addresses are redirected to the real server to keep the website operations for normal traffic. Thus, offending IP addresses are quarantined while normal requests are being served.

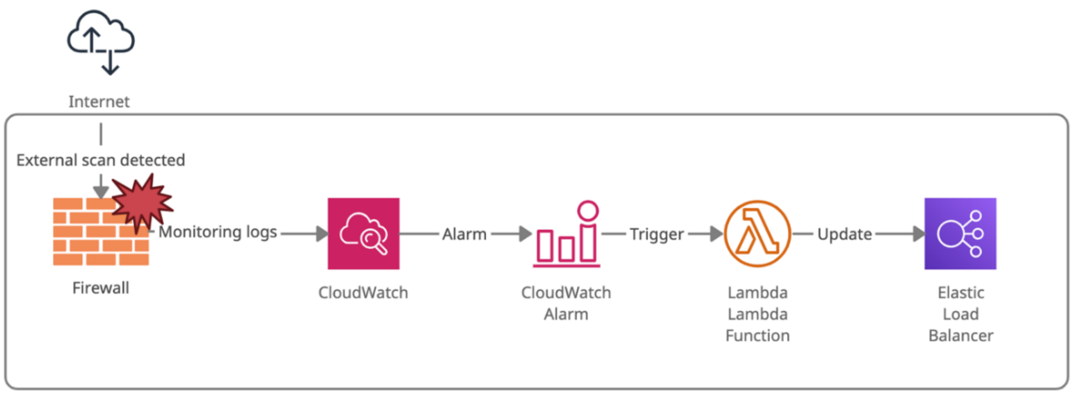


Figure 4 Elastic Load Balancer reconfiguration

Normal Mode and Defence Mode

In Normal Mode, only real hosts exist in the network, and no honeypot instances are provisioned. All incoming requests are forwarded to the webserver WS1. The oversensitivity of the hypothetical IDS generating false positives can be mitigated by using possible threats reported by the IDS as a trigger to enter Defence mode.

Dynamically provisioning honeypots in the cloud comes with some challenges. Network scan results of recently provisioned honeypots will reveal the server's uptime to the attacker unless uptime is obfuscated. Fast network scans also pose a threat: it is possible to scan a specific IP address and get results back before the honeypot instance is ready to respond to requests. Dynamic honeypot provisioning may not be fast enough to mitigate all types of network scanning. However, the attacker cannot differentiate between WS1 and WS2 because WS2 is a mirror copy of WS1. Even though an initial network scan might hit WS1, consecutive scans will only hit WS2 without the attacker being aware of the change in the background.

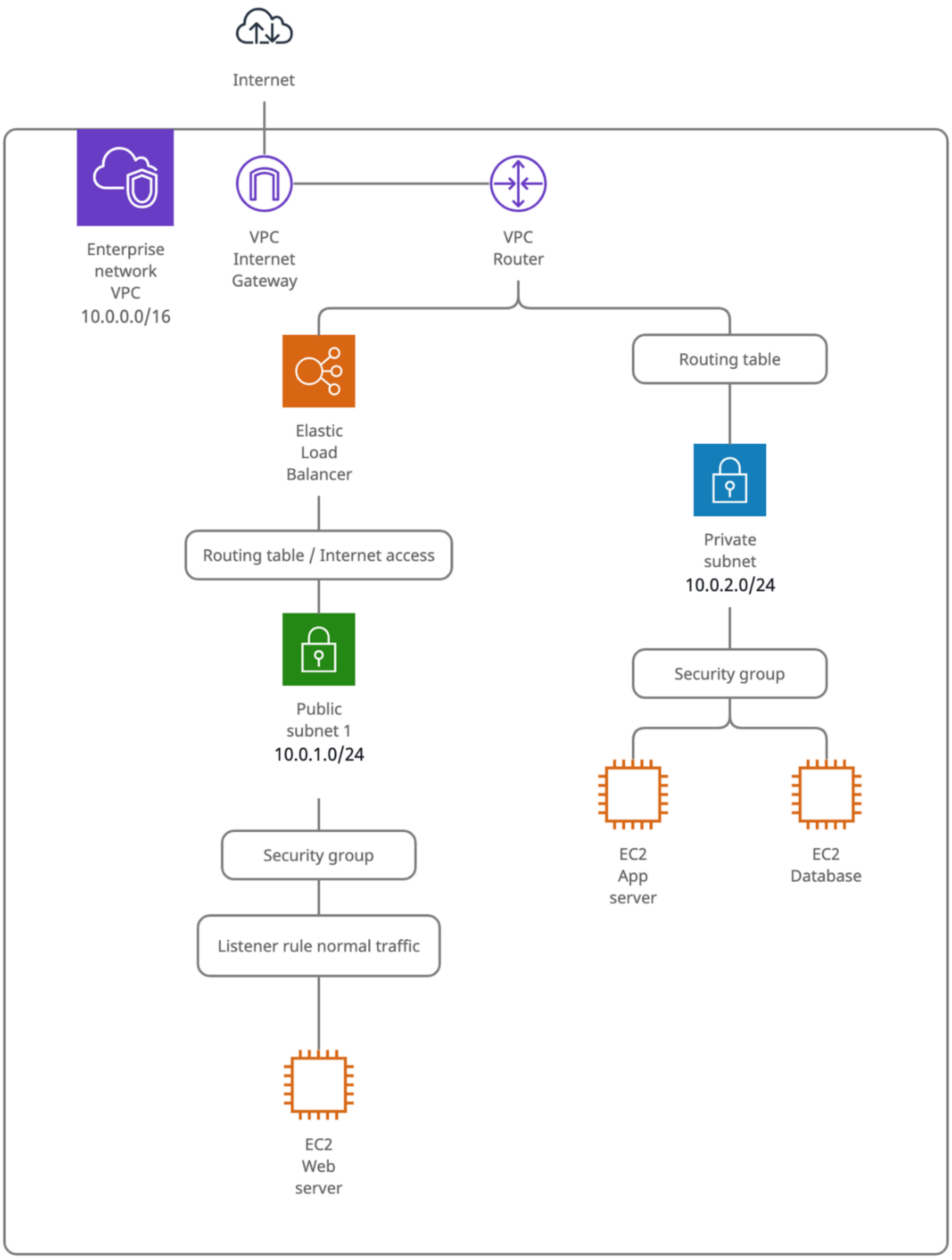

Internet
Enterprise network VPC 10.0.0.0/16
VPC Internet Gateway
VPC Router
Elastic Load Balancer
Routing table
Routing table / Internet access
Private subnet 10.0.2.0/24
Public subnet 1 10.0.1.0/24
Security group
Security group
EC2 App server
EC2 Database
Listener rule normal traffic
EC2 Web server

Figure 5 Enterprise Network architecture schematic in Normal Mode

The system goes into defence mode when a network anomaly is detected, such as a port scanning attempt. This event triggers the provisioning of honeypots from AMIs, adding them to the honeypot target group HTG and, subsequently, reconfiguring the WLB Elastic Load Balancer listener rule to redirect requests from the offending IP address to the HTG. WLB's listener rule reconfiguration must wait until the honeypots are available to serve requests and are part of the honeypot target group. Each new offending IP address is quarantined with a new listener rule. The system switches from Defence Mode to Normal Mode when no network anomaly is detected for a certain amount of time. The list of offending IPs may or may not be discarded depending on preference. Because IPV4 addresses are often reassigned to other clients, it is reasonable to discard quarantined IP addresses to allow legitimate clients to connect to WS1 in the future. In Normal Mode, the honeypot servers are shut down and terminated. By switching between Normal and Defence modes, honeypots servers only consume resources when required, reducing costs and resource consumption.

Defence Mode steps:

1. Network anomaly detected
2. Provision honeypot servers
3. Register honeypot instances to HTG
4. Create a new load balancer listener rule to quarantine the suspicious IP address
5. Requests coming from the quarantined IP address are being served by the honeypot web server target group HTG
6. Subsequent offending IP addresses also are quarantined
7. The real webserver is serving requests coming from non-offending IP addresses

Normal Mode steps:

1. No network anomaly is detected for the specified time threshold
2. Stop and terminate honeypot instances to save resources
3. Delete listener rule from WLB

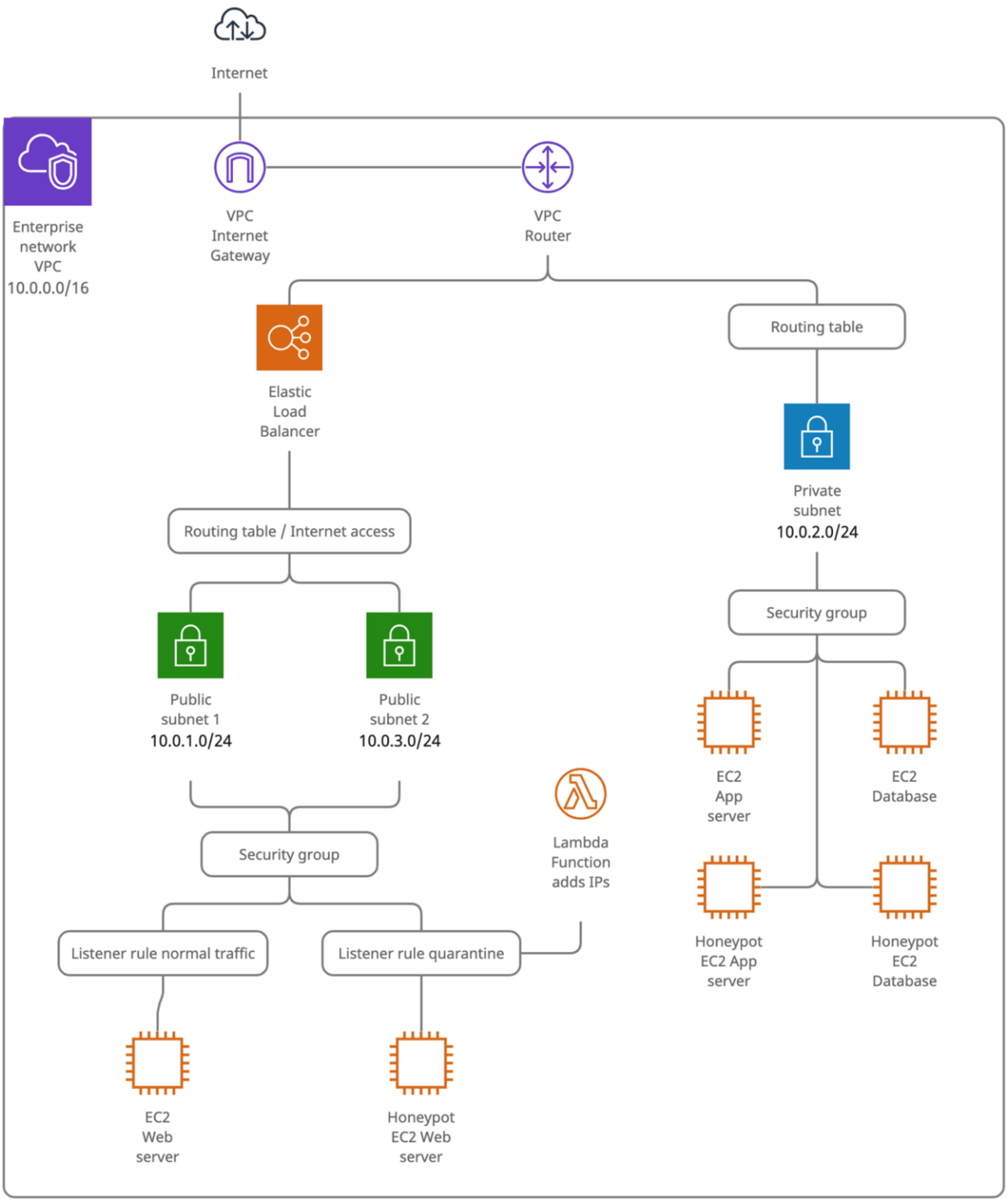


Figure 6 Enterprise Network architecture schematic in Defence mode

4. Tests

Tests are conducted using the AWS command-line interface (CLI) to prove the functionality of the system components, the believability of the provisioned honeypots, and the system's feasibility as a defence mechanism. AWS CLI is a command-line tool with JSON output for managing AWS resources. Command examples can be found in the AWS CLI Command Reference (Amazon Web Services, n.d.-c).

Test and component matrix

The purpose of the test matrix is to set up test scenarios for validating the functionality of each component, mapping measurable outcomes of the tests to the project objectives.

Table 9 Enterprise Network components matrix and test matrix

Resource Name	Resource Purpose	Network range	Target	Protocol
Enterprise network VPC	Virtual network	10.0.0.0/16		
Public subnet PS1	Subnet allowing internet access	10.0.1.0/24		
Public subnet PS2	Subnet allowing internet access	10.0.3.0/24		
Private subnet PRS	Subnet with no internet access	10.0.2.0/24	0.0.0.0/16	
Internet Gateway IG	Allows internet access to the Enterprise Network VPC	0.0.0.0/16	Enterprise network VPC	
Elastic Load Balancer WLB	Distributes incoming application traffic across EC2 instances	52.223.23.164, 35.71.153.62	Webserver EC2s	
Webserver Target Group WTG	The target group of webservers serving normal requests			
Honeypot Target Group HTG	The target group of honeypot webservers serving malicious traffic			
Routing table RT1	Defines network routes between the Internet Gateway and the public subnet	10.0.0.0/16 0.0.0.0/16	Local traffic Internet Gateway	
Routing table RT2	Defines network routes within the VPC for the private subnet	10.0.0.0/16	Local traffic	
Webserver Security Group WSG	Virtual Firewall to control inbound and outbound traffic to EC2s	0.0.0.0/16	Webserver EC2	HTTP 80, HTTPS 443 SSH 22
Database and Application server Security Group	Virtual Firewall to control inbound and outbound traffic to EC2s	10.0.2.0/24 0.0.0.0/16	Webserver Security Group Internet	MySQL 3306 RDP 3389
Webserver EC2 WS1	T2.micro Amazon Linux with Apache HTTP server		Internet-facing	HTTP HTTPS
Webserver EC2 WS2	T2.micro Amazon Linux with Apache HTTP server		Internet-facing	HTTP, HTTPS
Database server EC2 DB1	Windows Server with MySQL 5.7			
Application server EC2 AS1	Amazon Linux with PHP			
Lambda function start_ec2	Start existing EC instances			
Lambda function launch_ami	Create and start a new EC2 instance from AMI			
Lambda function HTG_update	To register an EC2 instance in target group HTG			
Lambda function ELB_update	To create a new load balancer listener rule for the ELB WLB for quarantining offending IP addresses			

Functional testing of honeypot provisioning				
ID	Test description	Test procedure	Expected output	Result: Pass
1.1	The test is designed to check if the Lambda function correctly starts the existing honeypot EC2 instance Web server WS2	Check initial instance status using AWS CLI Trigger Lambda function to start existing honeypot EC2 instance Web server WS2 and record timestamp Check if the EC2 instance has been started and record the timestamp Take a screenshot of the EC2 instance in the AWS console	Initial instance status is empty (stopped) Lambda function triggered successfully All instance status checks all pass Console screenshot showing running instance	Lambda function start_ec2 executes without error. WS2 EC2 instance starts successfully. WS2 is serving web requests. The average time until instance connectible is 40 seconds.
1.2	The test is designed to check if the Lambda function correctly creates and starts honeypot EC2 instance Web server WS2 from AMI	List existing instances Trigger Lambda function to create and start honeypot EC2 instance Web server WS2 from AMI and record timestamp Check if the EC2 instance has been created and is started in the EC2 console Check new instance status Check if httpd was started and try to open http://52.223.23.164/ repeatedly and record timestamp when hitting WS2	Timestamp of triggering Lambda function Timestamp of the server accepting SSH connections Timestamp of the server serving web requests Console screenshot of new instance running	Lambda function launch_ami executes without error. Webserver WS2 is created and started up successfully. The average time until instance connectible is 40 seconds.
Functional testing of Target Group reconfiguration				
ID	Test description	Test procedure	Expected output	Result: Pass
2.1	The test is designed to check if the Lambda function correctly updates the target group HTG and registers Web server WS2	Check if there are any targets registered in HTG Trigger lambda function HTG_update to register WS2 as a target in target group HTG Check if WS2 is registered in target group HTG using AWS CLI	Timestamp and console output of triggering Lambda function Timestamp and console output describing HTG including WS2 as target Screenshots from AWS console showing HTG including WS2 as target	The newly created web server is successfully registered as a target in the HTG target group.
Functional testing of Elastic Load Balancer reconfiguration				
ID	Test description	Test procedure	Expected output	Result: Pass
3.1	The test is designed to test if the Lambda function successfully and correctly updates the listener and creates a new rule for ELB WLB	Check current listener rule configuration Trigger Lambda function to update WLB listener rule to direct all requests from the test computer's public IP address to WS2 and record timestamp Check in the EC2 console if the listener rule has been updated and record timestamp Connect to VPN and open http://52.223.23.164/. Check if non-offending IP addresses are served by WS1 and not by WS2	Description of default listener rule configuration with timestamp Lambda function execution with timestamp Description of the listener with added listener rule forwarding the offending IP address to HTG Screenshot of opening http://52.223.23.164/ Screenshot of listener rule in AWS console	Lambda function ELB_update executes without error. A new listener rule was created with the offending IP address. When opening http://52.223.23.164/ on the test machine, responses are coming from WS2. Using a VPN, requests from non-offending IPs

				are served by WS1. The time to quarantine an IP address by creating a new listener rule is trivial.
Believability testing				
ID	Test description	Test procedure	Expected output	Result: Fail
4.1	The attacker conducts external network scanning using NMAP	Launch NMAP scan of http://52.223.23.164/ Test machine source IP address is quarantined Launch NMAP scan of http://52.223.23.164/	NMAP scan results for the two scans are the same	The two scans show different results due to the difference between Apache versions running on WS1 (2.4.48) and WS2 (2.4.33) webservers.
ID	Test description	Test procedure	Expected output	Result: Pass
4.2	The attacker conducts external network scanning using NMAP	Launch NMAP scan of http://52.223.23.164/ Test machine source IP address is quarantined Launch NMAP scan of http://52.223.23.164/	NMAP scan results for the two scans are the same	The two scans show the same result because the Apache versions on both WS1 and WS2 are the same (2.4.48)
4.3	The attacker conducts external network scanning using NMAP	Launch NMAP scan of http://52.223.23.164/ At the same time, quarantine test machine IP address Capture NMAP scan results	NMAP scan result shows scan hitting WS2	NMAP scan results show the scan hitting WS2 (visible from Apache version 2.4.33)
Timing tests				
ID	Test description	Test procedure	Expected output	Result: Pass
5.1	Capturing the time needed for honeypot EC2 instance WS2 to be reachable	Start an existing EC2 instance with start_ec2 Lambda function Record time when WS2 is accepting requests Create and start EC2 instance from AMI with launch_ami Lambda function Record time when WS2 is accepting requests	Timestamps showing the execution of Lambda scripts and WS2 website availability	The time required for an existing or new EC2 instance to start serving requests is under 40 seconds.

Test results

The test outcomes below are an abbreviated version of the full test output found in the appendix.

Test ID 1.1 is designed to check if the Lambda function start_ec2 correctly starts the existing honeypot EC2 instance Web server WS2

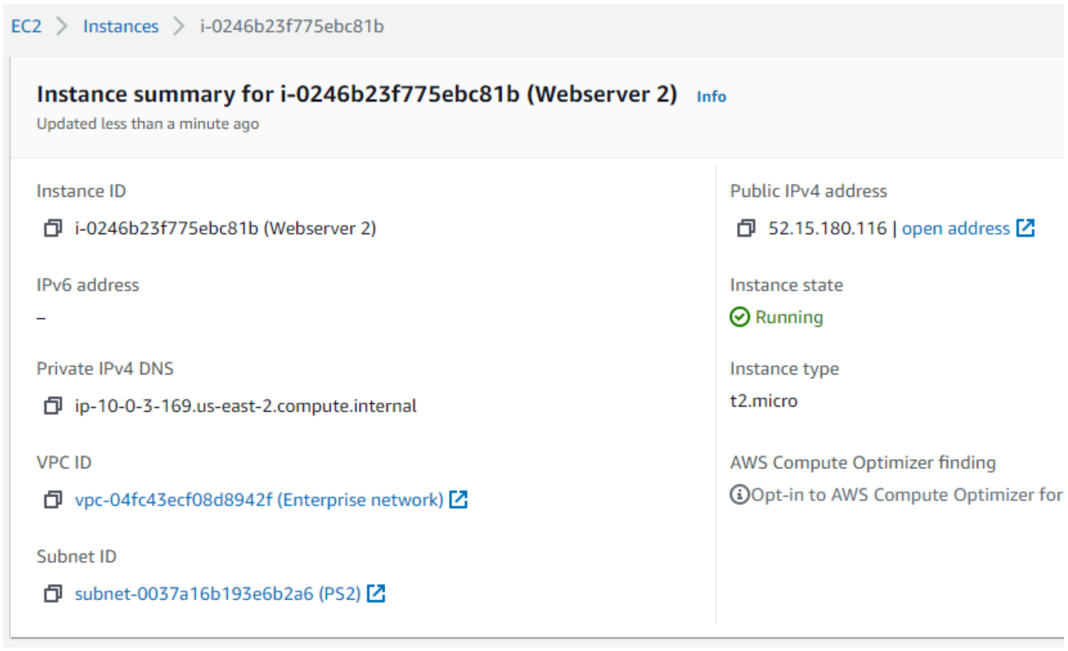
Test step	Test outcome	Test analysis

1. Checking initial instance status	`aws ec2 describe-instance-status --instance-id i-0246b23f775ebc81b`
	
	`"InstanceStatuses": []`
		The empty array returned, for instance, id i-0246b23f775ebc81b of InstanceStatuses confirms that the instance is not running.

2. Executing Lambda function start_ec2	`aws lambda invoke --function-name start_ec2`	Status code 200 is the successful execution of the Lambda function.

3. Checking instance status	`aws ec2 describe-instance-status --instance-id i-0246b23f775ebc81b`
	
	`"Name": "reachability",`
	`"Status": "passed"`
		After the Lambda function is executed, InstanceStatuses is populated with an availability zone, instance ID, state, and system statuses. These are built-in health checks of AWS.
		
		The instance does not need to pass these checks to be connectible or otherwise available.
		
		The check shows that all health checks pass.

Figure 7 AWS Console screenshot shows that the EC2 instance is now up and running.



EC2 > Instances > i-0246b23f775ebc81b

Instance summary for i-0246b23f775ebc81b (Webserver 2) Info

Updated less than a minute ago

Instance ID: i-0246b23f775ebc81b (Webserver 2)

Public IPv4 address: 52.15.180.116 | open address

IPv6 address: –

Instance state: Running

Private IPv4 DNS: ip-10-0-3-169.us-east-2.compute.internal

Instance type: t2.micro

VPC ID: vpc-04fc43ecf08d8942f (Enterprise network)

AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for

Subnet ID: subnet-0037a16b193e6b2a6 (PS2)

Test ID 1.2 is designed to check if the Lambda function launch_ami correctly creates and starts honeypot EC2 instance Web server WS2 from AMI

Test step	Test output	Test analysis

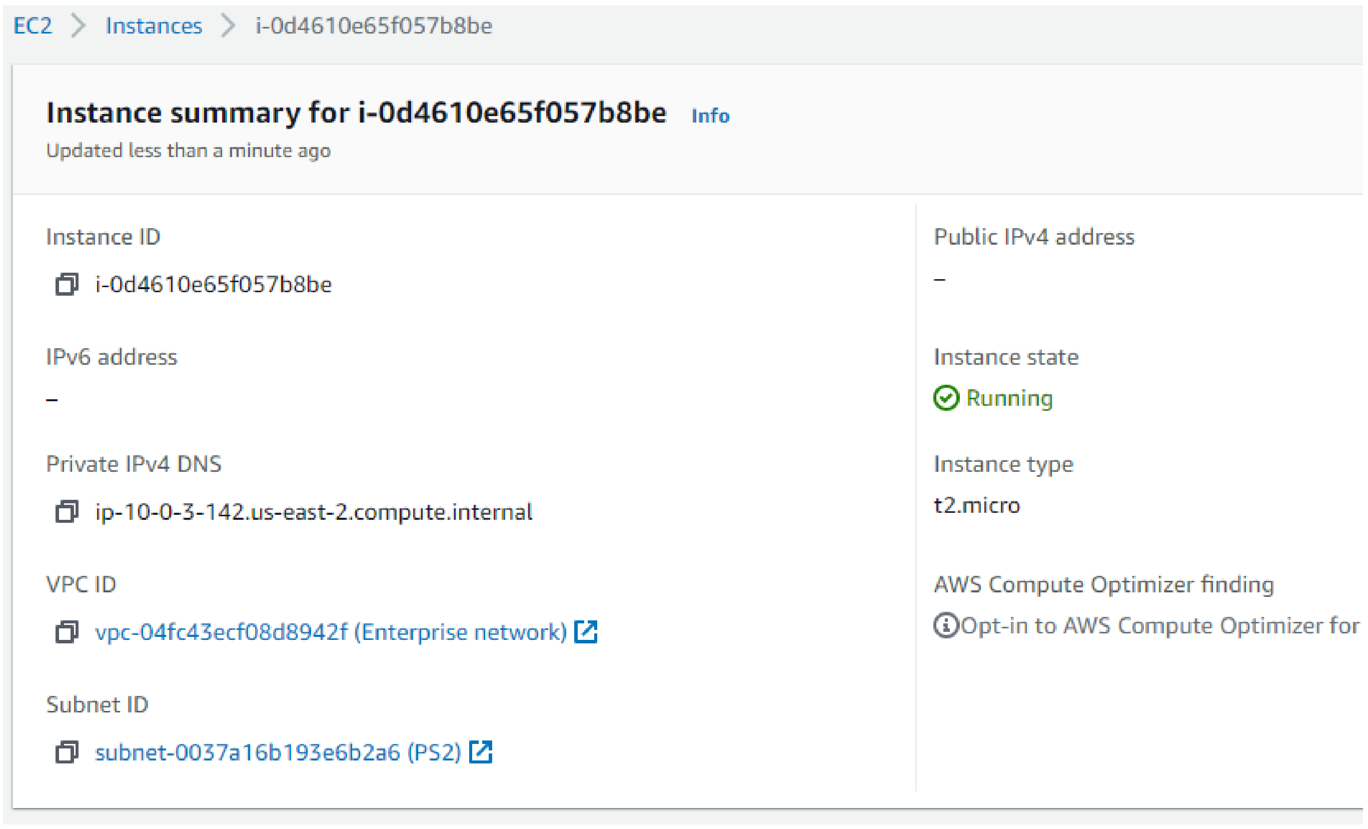
1. Listing existing instances	`aws ec2 describe-instances`
	
	`"Instance": "i-0c9623abffe6688aa",`
	`"Subnet": "subnet-0cadc5e13838fae4e"`
	`"Instance": "i-0246b23f775ebc81b",`
	`"Subnet": "subnet-0037a16b193e6b2a6"`
		The describe-instances command would return too much information about the EC instances, so a filter returns only the instance IDs and the associated subnets.

2. Executing Lambda function launch_ami	`lambda invoke --function-name launch_ami response.json`	Status code 200 is the successful execution of the Lambda function.

3. Checking if the new instance was created from the AMI	`aws ec2 describe-instances`
	
	`"Instance": "i-0d4610e65f057b8be",`
	`"Subnet": "subnet-0037a16b193e6b2a6"`
		The newly created instance ID i-0d4610e65f057b8be is listed.

4. Checking new instance status	`aws ec2 describe-instance-status --instance-id i-0d4610e65f057b8be`
	
	`"InstanceId": "i-0d4610e65f057b8be",`
	`"Status": "passed"`
		The new instance has been started, and AWS health checks passed.

Figure 8 AWS console screenshot shows that the newly created EC2 instance is now up and running.


EC2 > Instances > i-0d4610e65f057b8be

Instance summary for i-0d4610e65f057b8be Info

Updated less than a minute ago

Instance ID: i-0d4610e65f057b8be

Public IPv4 address: –

IPv6 address: –

Instance state: Running

Private IPv4 DNS: ip-10-0-3-142.us-east-2.compute.internal

Instance type: t2.micro

VPC ID: vpc-04fc43ecf08d8942f (Enterprise network)

AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for

Subnet ID: subnet-0037a16b193e6b2a6 (PS2)

Test ID 2.1 is designed to check if the Lambda function HTG_update registers Web server WS2 correctly.

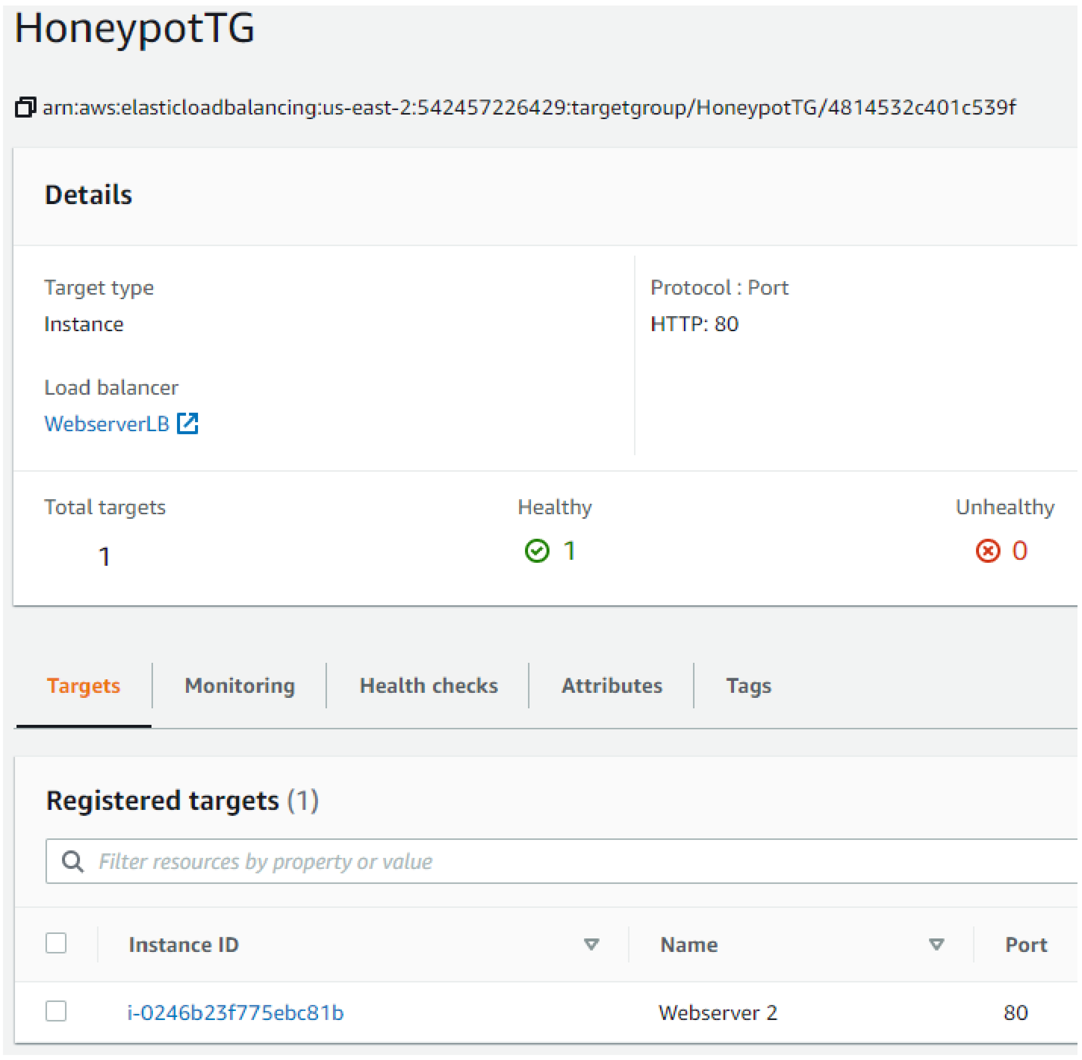
Test step	Test output	Test analysis

1. No targets registered in target group HTG	`aws elbv2 describe-target-health`
	
	`    "TargetHealthDescriptions": []`
		The empty array TargetHealthDescriptions returned shows that there are no targets registered in target group HTG

2. Executing Lambda function	`aws lambda invoke --function-name register-instance-to-HTG response.json`	Status code 200 is the successful execution of the Lambda function.

3. Checking if WS2 is registered in HTG	`aws elbv2 describe-target-health`
	
	`                "Description": "Target registration is in progress"`
	
	`aws elbv2 describe-target-health`
	
	`                 "State":  "healthy"`
		When the registration process has started and is underway, the target health check reports Elb.RegistrationInProgress.
		
		For the second check, the health check reports that the target is registered and is healthy.

Figure 9 AWS console screenshot of the updated HTG target group



HoneypotTG

arn:aws:elasticloadbalancing:us-east-2:542457226429:targetgroup/HoneypotTG/4814532c401c539f

Details

Target type	Protocol : Port
Instance	HTTP: 80
Load balancer	
WebserverLB	

Total targets	Healthy	Unhealthy
1	✓ 1	⊗ 0

Registered targets (1)

Instance ID	Name	Port
i-0246b23f775ebc81b	Webserver 2	80

Test ID 3.1 is designed to test if the Lambda function ELB_update successfully and correctly updates the listener for ELB WLB and creates a new listener rule forwarding requests from a specific IP address.

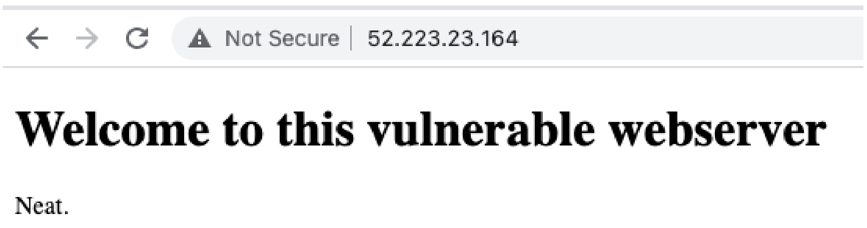
Test step	Test output	Test analysis

1. Checking current listener rule configuration	`aws elbv2 describe-rules`
	`"Actions": "Type": "forward",`
	`"ForwardConfig":`
	`"TargetGroupArn": "Webserver-target-group"`
		The describe-rules command lists the listener rules configured for the load balancer WLB. The single rule listed all requests forwards to the WTG target group, which has WS1 as the registered target.

2. Executing Lambda function ELB_update	`aws lambda invoke --function-name ELB_update`	Status code 200 is the successful execution of the Lambda function.

3. Checking if the listener rule has been updated with the correct configuration	`aws elbv2 describe-rules`
	`"SourceIpConfig":`
	`"Values": "87.80.156.133/32"`
	`"Actions": "Type": "forward",`
	`"TargetGroupArn": HoneypotTG`
		The new listener rule has been added to the load balancer listener configuration. The new rule is forwarding requests from the IP address range of 87.80.156.133/32 to the HTG target group where WS2 is registered as a target.

Figure 10 Screenshot of opening http://52.223.23.164/from quarantined IP address

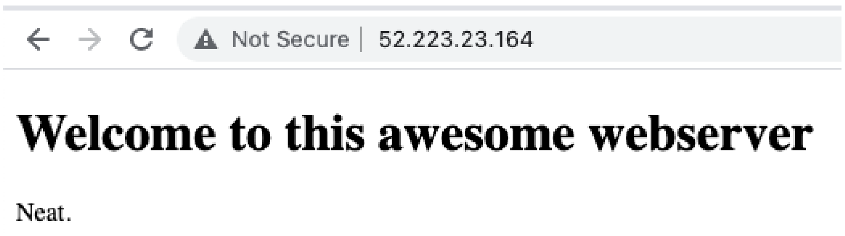

Not Secure | 52.223.23.164

Welcome to this vulnerable webserver

Neat.

The screenshot confirms that when visiting the website from a quarantined IP address, the request is forwarded by the load balancer to the WS2 honeypot web server based on the new listener rule.

Figure 11 Screenshot of non-quarantined IP address opening http://52.223.23.164/

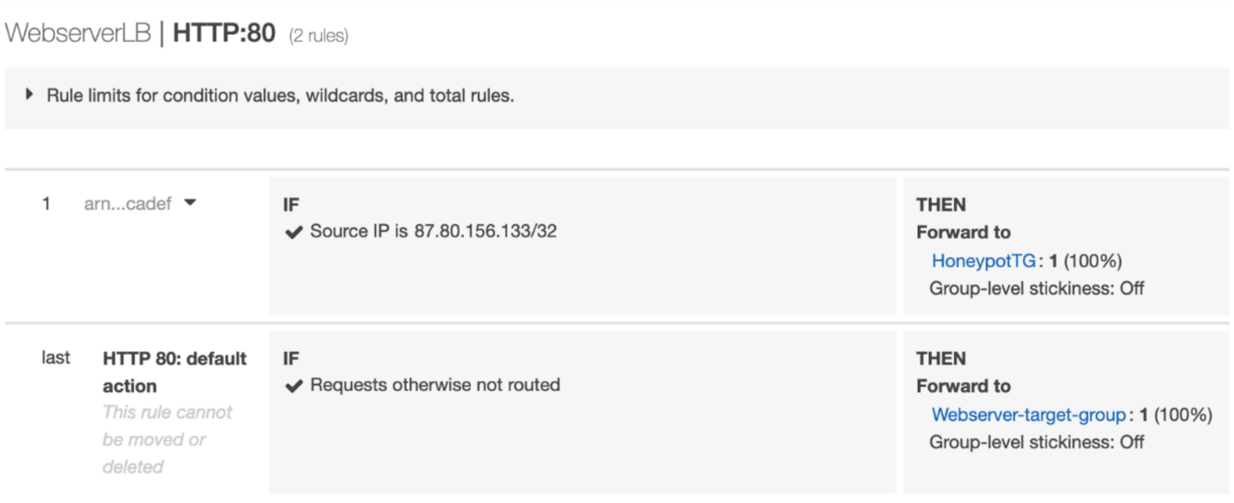

Not Secure | 52.223.23.164

Welcome to this awesome webserver

Neat.

The screenshot confirms that the load balancer forwards requests from non-offending IP addresses to the WTG target group where WS1 is the registered target.

Figure 12 Screenshot of the modified listener in the AWS console



WebserverLB | **HTTP:80** (2 rules)

▸ Rule limits for condition values, wildcards, and total rules.

		IF	THEN
1	arn...cadef ▾	✔ Source IP is 87.80.156.133/32	**Forward to** HoneypotTG: **1** (100%) Group-level stickiness: Off
last	**HTTP 80: default action** *This rule cannot be moved or deleted*	✔ Requests otherwise not routed	**Forward to** Webserver-target-group: **1** (100%) Group-level stickiness: Off

The screenshot confirms that the Lambda function ELB_update successfully created the new rule.

Test ID 4.1 is designed to test the believability of dynamic honeypot provisioning. The results of the two scans should be the same. This is not the case due to different software versions of Apache installed on WS1 and WS2.

Test step	Test output	Test analysis

1. NMAP scan in normal mode	`nmap -sV 52.223.23.164`
	`PORT   STATE SERVICE VERSION`
	`80/tcp open  http    Apache httpd 2.4.48`
		NMAP scan started from regular IP address hitting WS1

2. NMAP scan in defence mode	`nmap -sV 52.223.23.164`
	`PORT   STATE SERVICE VERSION`
	`80/tcp open  http    Apache httpd 2.4.33`
		NMAP scan started from quarantined IP address hitting WS2

Test ID 4.2 is the same test as ID 4.1 with matching Apache versions. If there is no difference in the test output, believability is achieved, the attacker cannot see any difference between WS1 and WS2.

Test step	Test output	Test analysis

1. NMAP scan in normal mode	`nmap -sV 52.223.23.164`
	`PORT   STATE SERVICE VERSION`
	`80/tcp open  http    Apache httpd 2.4.48`
		NMAP scan started from regular IP address hitting WS1

2. NMAP scan in defence mode	`nmap -sV 52.223.23.164`
	`PORT   STATE SERVICE VERSION`
	`80/tcp open  http    Apache httpd 2.4.48`
		NMAP scan started from quarantined IP address hitting WS2

Test ID 4.3 is designed to test the responsiveness of dynamic IP quarantining by the ways of a TCP port scan during which the IP address the scan was initiated from is quarantined. The test is successful because the results show WS2 responding to the scan, meaning it never hit WS1.

Test step	Test output	Test analysis
1. Start NMAP scan from non-quarantined IP address	`nmap -sV 52.223.23.164`	NMAP scan starts from regular IP address

2. At the same time as step 1, quarantine IP address by executing Lambda function ELB_update	`aws lambda invoke --function-name ELB_update respons`
	`e.json`
		Status code 200 is successful execution of the Lambda function.

3.Capture NMAP scan results	`PORT   STATE SERVICE VERSION`
	`80/tcp open  http    Apache httpd 2.4.33`
		NMAP scan results show scan result hitting WS2

Test ID 5.1 is designed to test the responsiveness of dynamic honeypot provisioning, capturing the timestamps of script executions and the website's availability. The first test is about starting the existing EC2 honeypot instance WS2 by executing the start_EC2 Lambda function. In contrast, the second test creates and starts a new honeypot EC2 instance using the launch_AMI Lambda function.

Test step	Test output	Test analysis
1.Stop WS2 EC2 instance		

1.Execute start_ec2 Lambda function	`aws lambda invoke --function-name start_ec2`
	`[ec2-user@ip-10-0-1-218 20210818-13:36:35]$`
		The Lambda function start_ec2 was executed at 13:36:35

2. Record timestamp when WS2 is accepting requests	`curl -Is http://35.71.153.62 | head -1`
	`HTTP/1.1 200 OK`
	`date`
	`Wed 18 Aug 2021 13:37:13`
		The website became available at 13:37:13. The time elapsed between executing the Lambda function start_ec2 until the website is available is 38 seconds.

3. Execute launch_ami Lambda function	`aws lambda invoke --function-name launch_ami`
	`[ec2-user@ip-10-0-1-218 20210818-14:31:35]$`
		Lambda function launch_ami was executed at 14:31:35

Execute HTG_update Lambda function	`aws lambda invoke --function-name HTG_update`
	`[ec2-user@ip-10-0-1-218 20210818-14:32:05]$`
		Lambda function HTG_update was executed at 14:32:05

Record timestamp when new EC2 webserver is accepting requests	`curl -Is http://35.71.153.62 | head -1`
	`HTTP/1.1 200 OK`
	`date`
	`Wed 18 Aug 2021 14:32:21`
		The website was available at 14:32:21. The time elapsed between executing the Lambda function launch_ami until the website is available in 46 seconds.

5. Analysis of test results

Functional test results

Functional testing proves that the system is operational and the objectives set out are achieved. Test ID 1.1 proves that EC2 instances can be started up successfully using Lambda function start_ec2. Test ID 1.2 proves that EC2 instances can be created from an AMI and started using Lambda function launch_ami. Honeypot instances are created and launched with pre-configured software.

Test ID 2.1 proves that target groups can be reconfigured by registering new targets using the Lambda function HTG_update. Test ID 3.1 proves that the Elastic Load Balancer ELB can be reconfigured by adding a new listener rule to forward requests from specific IP addresses using the Lambda function ELB_update.

Using serverless Lambda functions is a flexible and reliable way to programmatically reconfigure the infrastructure components based on dependencies of other AWS resources to switch between normal and defence modes. While the Lambda functions are separated for testing and logging purposes in this project, the functions can also be combined into a single Lambda function.

While this project focuses on novel, dynamic honeypot provisioning, it may be a good defence strategy to place static, low-interaction, always-on honeypots into the network for malware research purposes. Mixing static and dynamically provisioned honeypots may be a best-of-both-world combination.

Timing test results

Attackers must limit the speed of their port scans to avoid detection: network mapping and port scanning for larger networks with numerous hosts may take hours. Depending on the number of targets to scan and the type of scan performed, it can take up to 21 minutes for each host scanned (NMAP, n.d.). Scanning the single IP with NMAP -sV (version detection) took around 60 seconds. It takes around 40 seconds for a honeypot web server instance running Linux to be available and accept requests if the instance already exists but stopped or when the instance is created and started from an AMI. The systemd suite greatly reduces the boot speed of Linux servers, a system and service manager providing parallelisation for boot processes (Freedesktop.org, n.d.).

Test ID 5.1 proves that the objective of rapid honeypot provisioning is satisfied. There is no time difference between starting up an existing honeypot EC2 instance and creating and starting up a new EC2 instance from an AMI. In defence mode, the system can spin up new EC2 instances in under 40 seconds when malicious activity is detected. Rapid honeypot provisioning enables honeypots to spin up on-demand, only when needed. There is no need for honeypots to be up and running all the time, reducing emissions and hosting costs.

Startup times vary depending on the instance type, the type and number of software packages that need to be started automatically baked into the AMI, infrastructure load on AWS and the type of boot volume used. A database instance running on a Windows Server may take considerably longer to accept connections. For servers not facing the internet, boot time is of less importance.

Reconfiguring the Elastic Load Balancer's listener by adding a rule to forward requests from a specific IP address is instant. When defence mode is active, and honeypot instances are provisioned, new offending IP addresses are quarantined instantly.

The time it takes the honeypot resources to become available allows more checks to be run to evaluate and quarantine IP addresses if necessary.

Believability test analysis

Switching between Normal Mode and Defence Mode is quick and transparent to an attacker because there is no change in address resolution. Until honeypot resources are available, requests are forwarded to the real server. By provisioning the honeypot web server WS2 and reconfiguring the load balancer in under one minute, the objective of a quick response to malicious activity is achieved. The attacker will be diverted to interact with the fake web server, the honeypot, virtually indistinguishable from the real system, thus wasting their time and resources. The attacker is kept away from gathering information and enumerating targets on the real network.

Test results for test ID 4.1 reveal a possible issue with a software version mismatch. While the first scan shows Apache version 2.4.48 running on WS1, after defence mode is activated, subsequent scans will show Apache version 2.4.33 running on WS2. The difference in Apache versions could be a possible giveaway for the attacker. They may notice that they are now interacting with a different server, which may signal that their activities have been discovered. However, because the AWS global accelerator shows that the servers are behind a load balancer, the attacker may accept that the different servers are running different versions of Apache. The discovery of Apache version mismatch between load-balanced servers may reinforce the attacker in their pursuit of finding vulnerable services. It could also mean that the version mismatch would become a way for an attacker to test the effectiveness of the intrusion detection mechanism. Based on the different software versions, attackers can tell what activities are detected and, consequently, what activities are not detected. This may become a method to game the system and find activities that are not detected.

Thus, test ID 4.1 proves that the objective of providing honeypots indistinguishable from real systems is not achieved in this case. Revealing any information to attackers about defence mechanisms carries the possibility of using that information against the system. The issue can be mitigated by installing the same operating system and software versions on real and honeypot servers.

Initially, the project set out to provide a choice for the attacker and present slightly more vulnerable versions of real servers for honeypot servers as a lure. In the case of webservers, the load balancer takes the choice out of the equation. Thus there is no reason to capture and maintain vulnerable webserver AMIs. In the case of database and application servers, there is still a valid reason to present vulnerable versions of real servers to the attacker if they have breached the internal network. While relevant and interesting, internal network and port scanning are not in the scope of this project.

Test ID 4.2 repeats test ID 4.1 with matching Apache versions 2.4.48 running on both WS1 and WS2. Test ID 4.2 proves that the objective of provisioning honeypots as indistinguishable copies of real servers is achieved.

Test ID 4.3 proves that the objective of quarantining suspicious IP addresses during external network scanning is achieved. The NMAP scan is started before the offending IP is quarantined. The ELB_update Lambda function is executed simultaneously as the NMAP scan to update the Elastic Load Balancer ELB's listener configuration and add a forwarding rule for the test machine's IP address to be forwarded to WS2. By the time the scan is complete, the results show that WS2 is responding to the scan. Depending on the type of network scan, the system may be able to enter defence mode before the scan results are returned to the attacker. This way, the attacker does not gain information or interact with the real webserver WS1.

A possible issue with this system may be that, after some time, denial-of-service attacks may reveal to the attacker that they are interacting with a fake system. If a DoS attack is successful and puts a heavy load on the system, requests from quarantined IPs may not be served or maybe served considerably slower. Requests from regular IPs will still be served with normal response times by the real servers. This discrepancy may be mitigated by utilising auto-scaling. Provisioning more server resources into HTG would mimic the behaviour of a real system under a DoS attack.

Costs

AWS offers EC2 instances on-demand, spot instances and reserved instances. With on-demand, instances are paid for by the hour or by the second, depending on the instance type. On-demand is great for applications with unpredictable workloads that cannot be interrupted, such as honeypots. Spot instances are not suitable for honeypot deployment because while prices can be 90% lower when compared to on-demand pricing due to using unused AWS resources, capacity availability is unpredictable. Reserved instances offer 37%-57% savings when compared to on-demand instances with 1-3 years of commitment. Depending on the number of incidents from the IDS prompting Defence mode activations, reserved instances may be cheaper than on-demand instances.

Based on the prices in Table 9 below, the yearly on-demand price of a t3.xlarge honeypot instance running every day of the year for one hour would be 60.736 USD. Paying by the minute, the one-hour availability of the honeypot can be a single one-hour instantiation each day or sixty one-minute instantiations.

Table 10 AWS EC2 On-Demand pricing (*EC2 On-Demand Instance Pricing – Amazon Web Services*, n.d.)

Instance name	On-Demand hourly rate	vCPU	Memory	Storage	Network performance
a1.medium	$0.0255	1	2 GiB	EBS Only	Up to 10 Gigabit
a1.large	$0.051	2	4 GiB	EBS Only	Up to 10 Gigabit
a1.xlarge	$0.102	4	8 GiB	EBS Only	Up to 10 Gigabit
a1.2xlarge	$0.204	8	16 GiB	EBS Only	Up to 10 Gigabit
a1.4xlarge	$0.408	16	32 GiB	EBS Only	Up to 10 Gigabit
a1.metal	$0.408	16	32 GiB	EBS Only	Up to 10 Gigabit
t4g.nano	$0.0042	2	0.5 GiB	EBS Only	Up to 5 Gigabit
t4g.micro	$0.0084	2	1 GiB	EBS Only	Up to 5 Gigabit
t4g.small	$0.0168	2	2 GiB	EBS Only	Up to 5 Gigabit
t4g.medium	$0.0336	2	4 GiB	EBS Only	Up to 5 Gigabit
t4g.large	$0.0672	2	8 GiB	EBS Only	Up to 5 Gigabit
t4g.xlarge	$0.1344	4	16 GiB	EBS Only	Up to 5 Gigabit
t4g.2xlarge	$0.2688	8	32 GiB	EBS Only	Up to 5 Gigabit
t3.nano	$0.0052	2	0.5 GiB	EBS Only	Up to 5 Gigabit
t3.micro	$0.0104	2	1 GiB	EBS Only	Up to 5 Gigabit
t3.small	$0.0208	2	2 GiB	EBS Only	Up to 5 Gigabit
t3.medium	$0.0416	2	4 GiB	EBS Only	Up to 5 Gigabit
t3.large	$0.0832	2	8 GiB	EBS Only	Up to 5 Gigabit
t3.xlarge	$0.1664	4	16 GiB	EBS Only	Up to 5 Gigabit
t3.2xlarge	$0.3328	8	32 GiB	EBS Only	Up to 5 Gigabit

Environmental impact

Research commissioned by AWS shows that AWS's cloud infrastructure is 3.6 times more energy-efficient than regular data centres (451 Research, 2019). The biggest factor is the comparably higher server utilisation in the AWS infrastructure. AWS has an 88% reduced carbon footprint by using renewable energy sources compared to traditional data centres performing the same task.

Deploying hosts in cloud infrastructure can incur large savings both in costs and CO2 emissions. Cloud service providers offer shared tenancy and on-demand server provisioning to reduce costs and emissions compared to traditional data centre hosting. In the cloud, hosts are configured to start only when there is a need to do so, eliminating the over-provisioning of capacities. When demand is reduced, these hosts shut down automatically as defined by a set of rules. A well-architected infrastructure achieves the transition between levels of low and high demand seamlessly. Applying the same concept to honeypots, these hosts should also be only running when needed.

6. Conclusion

This project set out to design, build, and evaluate a novel method in dynamically deploying honeypots leveraging the opportunities offered by cloud hosting to overcome the shortcomings identified with current honeypot offerings. The project's objective was to increase honeypot believability and functionality and reduce maintenance time, costs, and emissions associated with running honeypot services. Cloud computing offers a range of previously untapped possibilities in how to deploy and use honeypots. Cloud computing allows networks and servers to be reconfigured on-demand with programmatically configurable virtual network and computing components.

Through successfully designing, building and testing the system, a novel concept is proposed how honeypots can be deployed in the cloud. Relying on non-existent resources to become available in reaction to an attack is a small but substantial original contribution that enhances honeypot functionality, reduces resource and maintenance costs and opens up new directions in which honeypots could be used in the future. The results are reproducible, and the design concept is transferrable to other cloud service providers.

Honeypot believability was achieved by building honeypot servers that are exact copies of real servers in the network. Network scans conducted from the attackers' perspective prove that honeypot versions of real servers created from AMIs are indistinguishable from real servers. Honeypot security was achieved by network segmentation. Requests from malicious IP addresses are forwarded to a separate subnet, isolating attackers from interacting with other critical subnets. The automation of dynamic honeypot deployment was achieved by using Lambda functions to provision and start honeypot instances on-demand. Honeypot availability was achieved by provisioning honeypots during a simulated attack promptly, using AMIs. The system only deploys honeypots when required. Energy and cost-saving objectives were achieved by relying on the on-demand aspects of provisioning computing resources in the cloud.

7. Further research opportunities

Cloud computing offers many ways to deploy honeypots and can change how we deploy honeypots and enhance honeypot functionality. In this section, some additional topics are touched upon that is out of this project's scope.

Full-stack honeypot provisioning

The test configuration in this project is limited to an Apache web server serving a static website. A full-stack configuration can be simulated by provisioning additional honeypot databases and application servers for maximum believability. In this case, data in the database will have to be anonymised to avoid leaking sensitive information in a successful data breach. The entire stack is duplicated in defence mode, serving requests running under the same configuration of slightly vulnerable software components. The honeypot web server, application and database servers are associated with their separate subnet to take advantage of network segmentation, virtually isolating the attacker from the rest of the Enterprise Network. This way, the entire application architecture is duplicated with anonymised data. Attackers are redirected and interact with the fake system, indistinguishable from the real system, in an isolated subnet. Such a system effectively wastes the attacker's resources and time, captures attack vector information for further threat analysis and an active defence system shielding the real application, and defends company assets at low maintenance and running costs.

Capturing threat analytics

Additional software may be installed on honeypots to discover and better understand novel attack strategies. Capturing attacker behaviour may aid in designing and building better defence systems.

Strategic honeypot positioning

Building on the current system, it may be possible to position honeypots in the network depending on the type and severity of an attack. By strategically positioning honeypots, attackers may be engaged longer and at a deeper level of interaction as the attack progresses.

Using Lambda function as a honeypot web server

It is possible to build serverless websites using Lambda in AWS. Lambda functions are the glue between AWS S3 for static content storage and DynamoDB, and the AWS managed NoSQL database system. This solution may further reduce costs and the need to provision, start and terminate honeypot servers using EC2. Furthermore, Lambda functions can be used as a target group member for the Elastic Load Balancer. However, it may require more maintenance, and believability is questionable as attackers will not be interacting with a copy of a real server.

References

Amazon Web Services, I. (n.d.-a). *Amazon EC2*. Retrieved August 9, 2021, from https://aws.amazon.com/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc

Amazon Web Services, I. (n.d.-b). *Amazon Virtual Private Cloud (VPC)*. Retrieved August 4, 2021, from https://aws.amazon.com/vpc/?vpc-blogs.sort-by=item.additionalFields.createdDate&vpc-blogs.sort-order=desc

Amazon Web Services, I. (n.d.-c). *AWS CLI Command Reference — AWS CLI 1.20.26 Command Reference*. Retrieved August 23, 2021, from https://docs.aws.amazon.com/cli/latest/index.html

Amazon Web Services, I. (n.d.-d). *Elastic Load Balancing - Amazon Web Services*. Retrieved August 4, 2021, from https://aws.amazon.com/elasticloadbalancing/?whats-new-cards-elb.sort-by=item.additionalFields.postDateTime&whats-new-cards-elb.sort-order=desc

Amazon Web Services, I. (n.d.-e). *Global Infrastructure Regions & AZs*. Retrieved August 12, 2021, from https://aws.amazon.com/about-aws/global-infrastructure/regions_az/

Amazon Web Services, I. (n.d.-f). *Internet gateways - Amazon Virtual Private Cloud*. Retrieved August 21, 2021, from https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Internet_Gateway.html

Amazon Web Services, I. (n.d.-g). *Quickstart — Boto3 Docs 1.18.19 documentation*. Retrieved August 12, 2021, from https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html

Amazon Web Services, I. (n.d.-h). *Security groups for your VPC - Amazon Virtual Private Cloud*. Retrieved August 9, 2021, from https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html

Amazon Web Services, I. (n.d.-i). *VPCs and subnets - Amazon Virtual Private Cloud*. Retrieved August 18, 2021, from https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Subnets.html#vpc-subnet-basics

Baykara, M., & Das, R. (2018). A novel honeypot based security approach for real-time intrusion detection and prevention systems. *Journal of Information Security and Applications*, *41*, 103–116. https://doi.org/10.1016/j.jisa.2018.06.004

Deception Logic Inc. (n.d.). *HoneyDB*. Retrieved March 24, 2021, from https://honeydb.io/about

EC2 On-Demand Instance Pricing – Amazon Web Services. (n.d.). Retrieved August 21, 2021, from https://aws.amazon.com/ec2/pricing/on-demand/

Freedesktop.org. (n.d.). *systemd*. Retrieved August 18, 2021, from https://www.freedesktop.org/wiki/Software/systemd/

Fu, X., Yu, W., Cheng, D., Tan, X., Streff, K., & Graham, S. (2006). On recognizing virtual honeypots and countermeasures. *Proceedings - 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing, DASC 2006*. https://doi.org/10.1109/DASC.2006.36

Jafarian, J. H., & Niakanlahiji, A. (2020). Delivering Honeypots as a Service. *Proceedings of the 53rd Hawaii International Conference on System Sciences*. https://doi.org/10.24251/hicss.2020.227

Kambow, N., & Passi, L. K. (2014). Honeypots : The Need of Network Security. *International*

Journal of Computer Science and Information Technologies, *5*(5).

lyrebird. (n.d.). *lyrebird/honeypot-base - Docker Image | Docker Hub*. Retrieved August 21, 2021, from https://hub.docker.com/r/lyrebird/honeypot-base/

MITRE. (n.d.). *Apache Http Server version 2.4.33 : Security vulnerabilities*. Retrieved August 17, 2021, from https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-595392/Apache-Http-Server-2.4.33.html

Naik, N., Jenkins, P., Cooke, R., & Yang, L. (2018). Honeypots that bite back: A fuzzy technique for identifying and inhibiting fingerprinting attacks on low interaction honeypots. *IEEE International Conference on Fuzzy Systems*, *2018-July*. https://doi.org/10.1109/FUZZ-IEEE.2018.8491456

NMAP. (n.d.). *Coping Strategies for Long Scans | Nmap Network Scanning*. Retrieved August 17, 2021, from https://nmap.org/book/scantime-coping.html

Peter, E., & Schiller, T. (2008). *A Practical Guide to Honeypots*. https://www.cse.wustl.edu/~jain/cse571-09/ftp/honey/#sec1.4

R0hi7. (n.d.). *GitHub - r0hi7/HoneySMB: Simple High Interaction Honeypot Solution for SMB protocol*. Retrieved August 21, 2021, from https://github.com/r0hi7/HoneySMB

Research 451. (2019). *The Carbon Reduction Opportunity of Moving to Amazon Web Services*. https://sustainability.aboutamazon.com/carbon_reduction_aws.pdf

Scott, S. (n.d.). *AWS global infrastructure, region table, data center location, availability*. Retrieved August 21, 2021, from https://cloudacademy.com/blog/aws-global-infrastructure/

Tsikerdekis, M., Zeadally, S., Schlesener, A., & Sklavos, N. (2019, February 5). Approaches for Preventing Honeypot Detection and Compromise. *2018 Global Information Infrastructure and Networking Symposium, GIIS 2018*. https://doi.org/10.1109/GIIS.2018.8635603

Appendix

Complete test results

Test ID 1.1

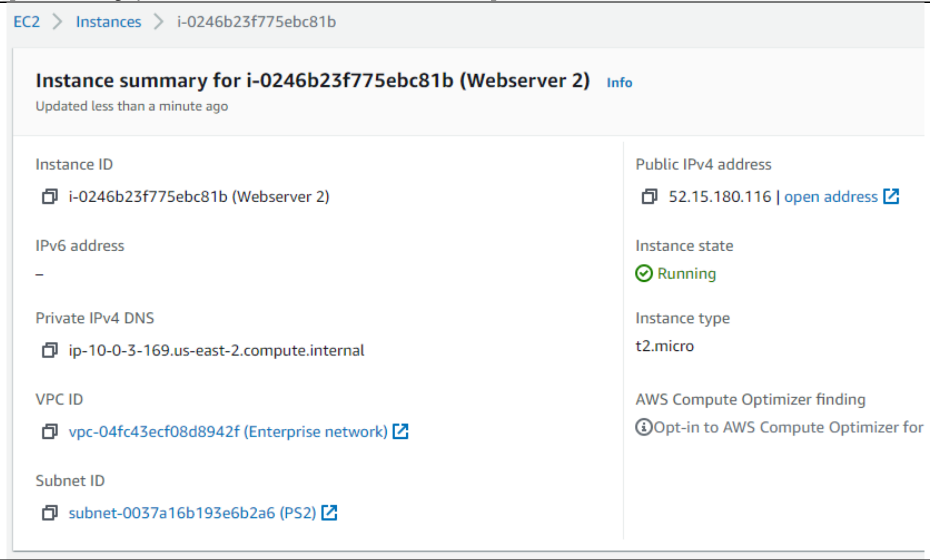
Test step	Test output	Test analysis

1. Checking initial instance status	`aws ec2 describe-instance-status --instance-id i-0246b23f775ebc81b`
	
	`{`
	`    "InstanceStatuses": []`
	`}`
		The empty array returned, for instance, id i-0246b23f775ebc81b of InstanceStatuses confirms that the instance is not running.

2. Executing Lambda function start_ec2	`[ec2-user@ip-10-0-1-218 20210817-09:15:30]$ aws lambda invoke --function-name s`
	`tart_ec2 response.json`
	`{`
	`    "StatusCode": 200,`
	`    "ExecutedVersion": "$LATEST"`
	`}`
		Status code 200 is the successful execution of the Lambda function.

3. Checking instance status	`[ec2-user@ip-10-0-1-218 20210817-09:16:40]$ aws ec2 describe-instance-status --`
	`instance-id i-0246b23f775ebc81b`
	`{`
	`    "InstanceStatuses": [`
	`        {`
	`            "AvailabilityZone": "us-east-2b",`
	`            "InstanceId": "i-0246b23f775ebc81b",`
	`            "InstanceState": {`
	`                "Code": 16,`
	`                "Name": "running"`
	`            },`
	`            "InstanceStatus": {`
	`                "Details": [`
	`                    {`
	`                        "Name": "reachability",`
	`                        "Status": "initializing"`
	`                    }`
	`                ],`
	`                "Status": "initializing"`
	`            },`
	`            "SystemStatus": {`
	`                "Details": [`
	`                    {`
	`                        "Name": "reachability",`
	`                        "Status": "initializing"`
	`                    }`
	`                ],`
	`                "Status": "initializing"`
	`            }`
	`        }`
	`    ]`
	`}`
	Check repeated
	
	`[ec2-user@ip-10-0-1-218 20210817-09:17:20]$ aws ec2 describe-instance-status --`
	`instance-id i-0246b23f775ebc81b`
	`{`
	`    "InstanceStatuses": [`
	`        {`
	`            "AvailabilityZone": "us-east-2b",`
	`            "InstanceId": "i-0246b23f775ebc81b",`
	`            "InstanceState": {`
	`                "Code": 16,`
	`                "Name": "running"`
	`            },`
	`            "InstanceStatus": {`
		After the Lambda function is executed, InstanceStatuses is populated with an availability zone, instance ID, state, and system statuses. These are built-in health checks of AWS.
		
		The instance does not need to pass these checks to be connectible or otherwise available.
		
		The second check shows that all health checks pass at 09:17:20.

	`                "Details": [`
	`                    {`
	`                        "Name": "reachability",`
	`                        "Status": "passed"`
	`                    }`
	`                ],`
	`                "Status": "ok"`
	`            },`
	`            "SystemStatus": {`
	`                "Details": [`
	`                    {`
	`                        "Name": "reachability",`
	`                        "Status": "passed"`
	`                    }`
	`                ],`
	`                "Status": "ok"`
	`            }`
	`        }`
	`    ]`
	`}`
	`[ec2-user@ip-10-0-1-218 20210817-09:18:33]$`	

4. AWS console screenshot	 EC2 > Instances > i-0246b23f775ebc81b **Instance summary for i-0246b23f775ebc81b (Webserver 2)** Info Updated less than a minute ago Instance ID: i-0246b23f775ebc81b (Webserver 2) Public IPv4 address: 52.15.180.116 | open address IPv6 address: – Instance state: Running Private IPv4 DNS: ip-10-0-3-169.us-east-2.compute.internal Instance type: t2.micro VPC ID: vpc-04fc43ecf08d8942f (Enterprise network) AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for Subnet ID: subnet-0037a16b193e6b2a6 (PS2)	The screenshot shows that the EC2 instance is now up and running.

Test ID 1.2

Test step	Test output	Test analysis

1. Listing existing instances	`[ec2-user@ip-10-0-1-218 20210817-09:34:16]$ aws ec2 describe-instances --query`
	`'Reservations[*].Instances[*].{Instance:InstanceId,Subnet:SubnetId}' --output j`
	`son`
	`[`
	`    [`
	`        {`
	`            "Instance": "i-0c9623abffe6688aa",`
	`            "Subnet": "subnet-0cadc5e13838fae4e"`
	`        }`
	`    ],`
	`    [`
	`        {`
	`            "Instance": "i-0246b23f775ebc81b",`
	`            "Subnet": "subnet-0037a16b193e6b2a6"`
	`        }`
	`    ]`
	`]`
		The describe-instances command would return too much information about the EC instances, so a filter is used to return only the instance IDs and the associated subnets.

2. Executing Lambda function launch_ami	`[ec2-user@ip-10-0-1-218 20210817-09:35:28]$ aws lambda invoke --function-name l`
	`aunch_ami response.json`
	`{`
	`    "StatusCode": 200,`
	`    "ExecutedVersion": "$LATEST"`
	`}`
	`[ec2-user@ip-10-0-1-218 20210817-09:35:33]$`
		Status code 200 is the successful execution of the Lambda function.

3. Checking if the new instance was created from the AMI

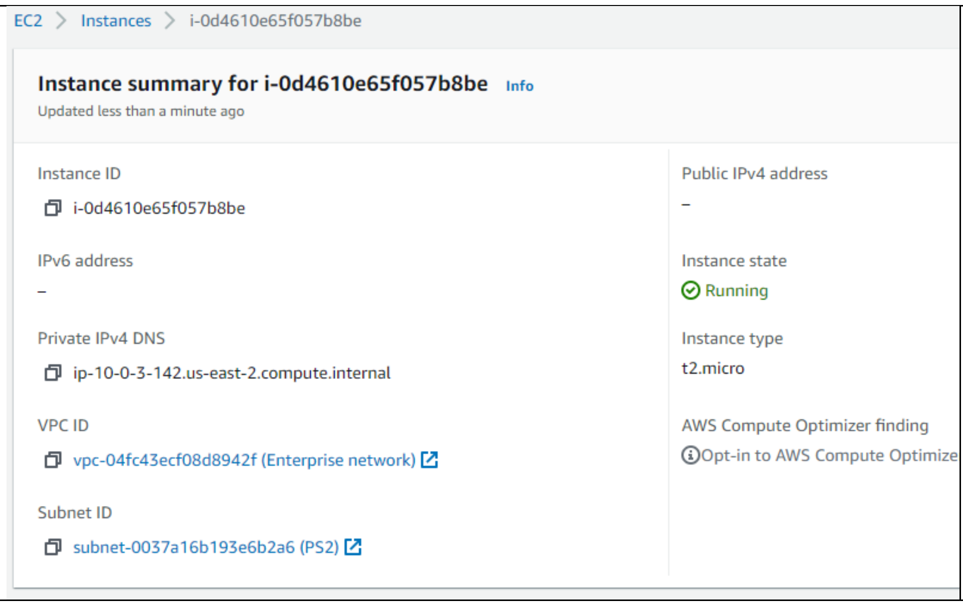
```
[ec2-user@ip-10-0-1-218 20210817-09:35:35]$ aws ec2 describe-instances --query
'Reservations[*].Instances[*].{Instance:InstanceId,Subnet:SubnetId}' --output j
son

[
    [
        {
            "Instance": "i-0c9623abffe6688aa",
            "Subnet": "subnet-0cadc5e13838fae4e"
        }
    ],
    [
        {
            "Instance": "i-0246b23f775ebc81b",
            "Subnet": "subnet-0037a16b193e6b2a6"
        }
    ],
    [
        {
            "Instance": "i-0d4610e65f057b8be",
            "Subnet": "subnet-0037a16b193e6b2a6"
        }
    ]
]
```

The newly created instance ID i-0d4610e65f057b8be is listed.

4. Checking new instance status

```
[ec2-user@ip-10-0-1-218 20210817-09:37:34]$ aws ec2 describe-instance-status --
instance-id i-0d4610e65f057b8be
{
    "InstanceStatuses": [
        {
            "AvailabilityZone": "us-east-2b",
            "InstanceId": "i-0d4610e65f057b8be",
            "InstanceState": {
                "Code": 16,
                "Name": "running"
            },
            "InstanceStatus": {
                "Details": [
                    {
                        "Name": "reachability",
                        "Status": "passed"
                    }
                ],
                "Status": "ok"
            },
            "SystemStatus": {
                "Details": [
                    {
                        "Name": "reachability",
                        "Status": "passed"
                    }
                ],
                "Status": "ok"
            }
        }
    ]
}
```

The new instance has been started, and AWS health checks are passed.

4. AWS console screenshot	 EC2 > Instances > i-0d4610e65f057b8be **Instance summary for i-0d4610e65f057b8be** Info Updated less than a minute ago Instance ID: i-0d4610e65f057b8be Public IPv4 address: – IPv6 address: – Instance state: Running Private IPv4 DNS: ip-10-0-3-142.us-east-2.compute.internal Instance type: t2.micro VPC ID: vpc-04fc43ecf08d8942f (Enterprise network) AWS Compute Optimizer finding: Opt-in to AWS Compute Optimize Subnet ID: subnet-0037a16b193e6b2a6 (PS2)	The screenshot shows that the newly created EC2 instance is now up and running. This confirms that the Lambda function has successfully created and started an instance from the AMI.

Test ID 2.1

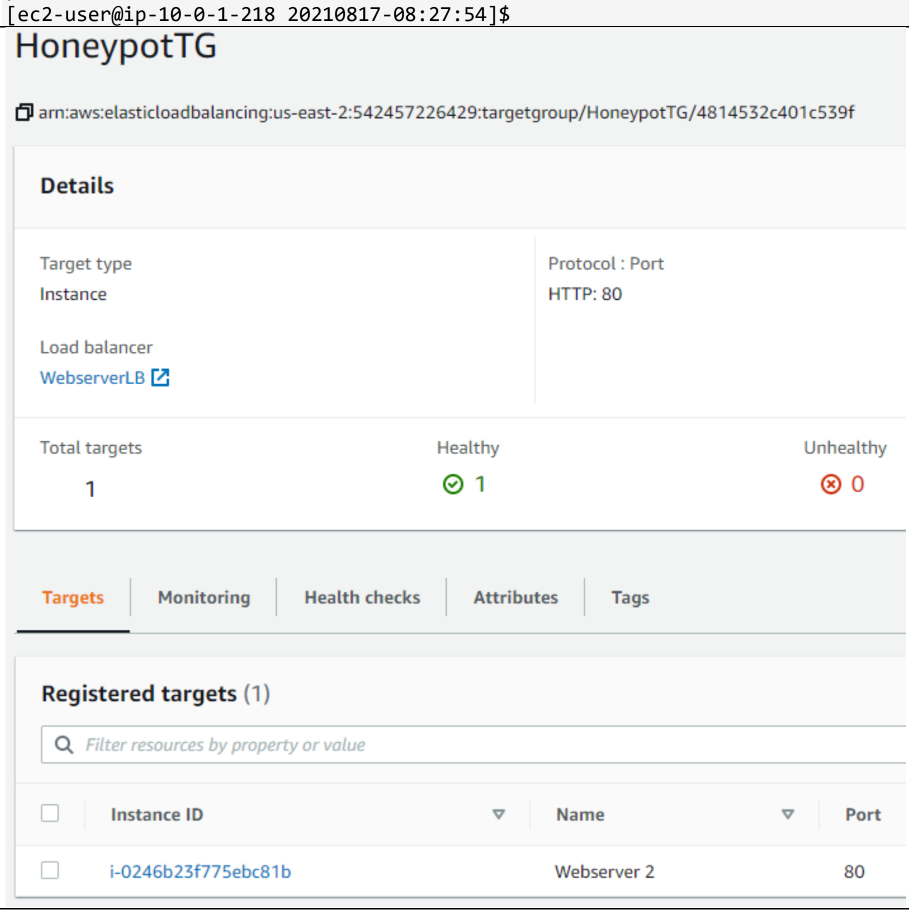
Test step	Test output	Test analysis

1. No targets registered in target group HTG	`aws elbv2 describe-target-health --target-group-arn arn:aws:elasticloadbalancing:`
	`us-east-2:542457226429:targetgroup/HoneypotTG/4814`
	`532c401c539f`
	`{`
	`    "TargetHealthDescriptions": []`
	`}`
	`[ec2-user@ip-10-0-1-218 20210817-08:26:42]$`
		The empty array TargetHealthDescriptions returned shows that there are no targets registered in target group HTG

2. Executing Lambda function	`[ec2-user@ip-10-0-1-218 20210817-08:27:36]$ aws lambda invoke --function-name reg`
	`ister-instance-to-HTG response.json`
	`{`
	`    "StatusCode": 200,`
	`    "ExecutedVersion": "$LATEST"`
	`}`
		Status code 200 is the successful execution of the Lambda function.

3. Checking if WS2 is registered in HTG	`ec2-user@ip-10-0-1-218 20210817-08:27:39]$ aws elbv2 describe-target-health --tar`
	`get-group-arn arn:aws:elasticloadbalancing:us-east-2:542457226429:targetgroup/Hon`
	`eypotTG/4814`
	`532c401c539f`
	`{`
	`    "TargetHealthDescriptions": [`
	`        {`
	`            "Target": {`
	`                "Id": "i-0246b23f775ebc81b",`
	`                "Port": 80`
	`            },`
	`            "HealthCheckPort": "80",`
	`            "TargetHealth": {`
	`                "State": "initial",`
	`                "Reason": "Elb.RegistrationInProgress",`
	`                "Description": "Target registration is in progress"`
	`            }`
	`        }`
	`    ]`
	`}`
	Checking again
	`[ec2-user@ip-10-0-1-218 20210817-08:27:49]$ aws elbv2 describe-target-health --ta`
	`rget-group-arn arn:aws:elasticloadbalancing:us-east-2:542457226429:targetgroup/Ho`
	`neypotTG/4814`
	`532c401c539f`
	`{`
	`    "TargetHealthDescriptions": [`
	`        {`
		When the registration process has started and is underway, the target health check reports Elb.RegistrationInProgress.
		
		For the second check, the health check reports that the target is registered and is healthy.

```
            "Target": {
                "Id": "i-0246b23f775ebc81b",
                "Port": 80
            },
            "HealthCheckPort": "80",
            "TargetHealth": {
                "State": "healthy"
            }
        }
    ]
}
[ec2-user@ip-10-0-1-218 20210817-08:27:54]$
```

4. AWS console screenshot		The screenshot confirms that the Lambda function has successfully registered the target into the target group HTG and is ready as the target for the load balancer.

HoneypotTG

arn:aws:elasticloadbalancing:us-east-2:542457226429:targetgroup/HoneypotTG/4814532c401c539f

Details

Target type: Instance

Protocol : Port: HTTP: 80

Load balancer: WebserverLB

Total targets: 1 | Healthy: 1 | Unhealthy: 0

Registered targets (1)

Instance ID	Name	Port
i-0246b23f775ebc81b	Webserver 2	80

Test ID 3.1

Test step	Test output	Test analysis
1. Checking current listener rule configuration	(output below)	The describe-rules command lists the listener rules configured for the load balancer WLB. The single rule listed all requests forwards to the WTG target group, which has WS1 as the registered target.
2. Executing Lambda function ELB_update	(output below)	Status code 200 is the successful execution of the Lambda function.
3. Checking if the listener rule has been updated with the correct configuration	(output below)	The new listener rule has been added to the load balancer listener configuration. The new rule is forwarding requests from the IP address range of 87.80.156.133/32 to the HTG target group where WS2 is registered as target.

1. Checking current listener rule configuration

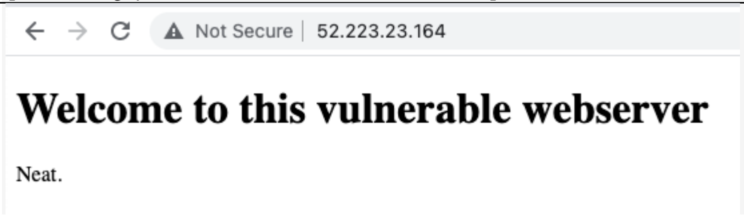
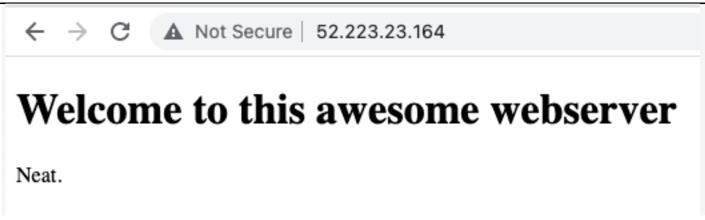
```
[ec2-user@ip-10-0-1-218 20210817-13:08:48]$ aws elbv2 describe-rules     --listen
er-arn arn:aws:elasticloadbalancing:us-east-2:542457226429:listener/app/Webserver
LB/5a2b7a5bf786ca1e/a82ef0acaacf173b
{
    "Rules": [
        {
            "RuleArn": "arn:aws:elasticloadbalancing:us-east-2:542457226429:liste
ner-rule/app/WebserverLB/5a2b7a5bf786ca1e/a82ef0acaacf173b/2e5e8e7cf1738582",
            "Priority": "default",
            "Conditions": [],
            "Actions": [
                {
                    "Type": "forward",
                    "TargetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:542
457226429:targetgroup/Webserver-target-group/9d25ac05032a4a25",
                    "ForwardConfig": {
                        "TargetGroups": [
                            {
                                "TargetGroupArn": "arn:aws:elasticloadbalancing:u
s-east-2:542457226429:targetgroup/Webserver-target-group/9d25ac05032a4a25",
                                "Weight": 1
                            }
                        ],
                        "TargetGroupStickinessConfig": {
                            "Enabled": false
                        }
                    }
                }
            ],
            "IsDefault": true
        }
    ]
}
```

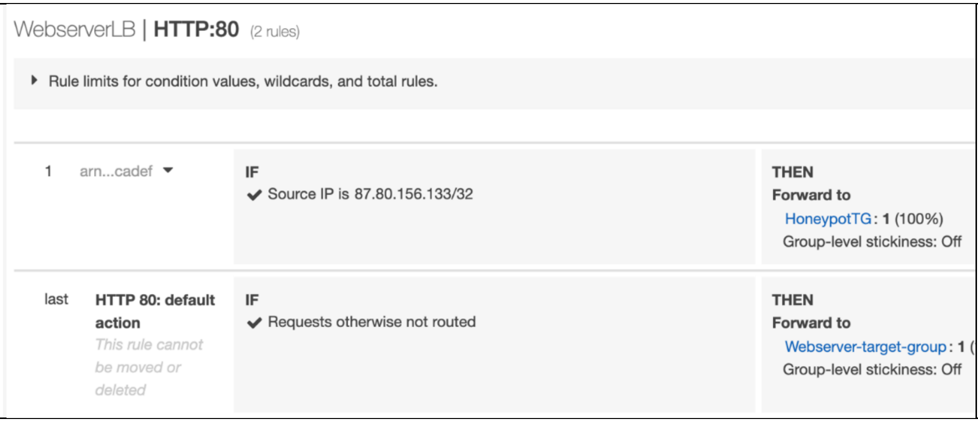
2. Executing Lambda function ELB_update

```
[ec2-user@ip-10-0-1-218 20210817-13:08:50]$ aws lambda invoke --function-name ELB
_update response.json
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

3. Checking if the listener rule has been updated with the correct configuration

```
[ec2-user@ip-10-0-1-218 20210817-13:14:05]$ aws elbv2 describe-rules     --listen
er-arn arn:aws:elasticloadbalancing:us-east-2:542457226429:listener/app/Webserver
LB/5a2b7a5bf786ca1e/a82ef0acaacf173b
{
    "Rules": [
        {
            "RuleArn": "arn:aws:elasticloadbalancing:us-east-2:542457226429:liste
ner-rule/app/WebserverLB/5a2b7a5bf786ca1e/a82ef0acaacf173b/ae58f12ebb7cadef",
            "Priority": "10",
            "Conditions": [
                {
                    "Field": "source-ip",
                    "SourceIpConfig": {
                        "Values": [
                            "87.80.156.133/32"
                        ]
                    }
                }
            ],
            "Actions": [
                {
                    "Type": "forward",
                    "TargetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:542
457226429:targetgroup/HoneypotTG/4814532c401c539f",
                    "ForwardConfig": {
                        "TargetGroups": [
                            {
                                "TargetGroupArn": "arn:aws:elasticloadbalancing:u
s-east-2:542457226429:targetgroup/HoneypotTG/4814532c401c539f",
                                "Weight": 1
                            }
                        ],
                        "TargetGroupStickinessConfig": {
                            "Enabled": false
                        }
                    }
                }
            ],
            "IsDefault": false
        },
        {
            "RuleArn": "arn:aws:elasticloadbalancing:us-east-2:542457226429:liste
ner-rule/app/WebserverLB/5a2b7a5bf786ca1e/a82ef0acaacf173b/2e5e8e7cf1738582",
            "Priority": "default",
            "Conditions": [],
            "Actions": [
                {
                    "Type": "forward",
                    "TargetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:542
457226429:targetgroup/Webserver-target-group/9d25ac05032a4a25",
                    "ForwardConfig": {
                        "TargetGroups": [
                            {
                                "TargetGroupArn": "arn:aws:elasticloadbalancing:u
s-east-2:542457226429:targetgroup/Webserver-target-group/9d25ac05032a4a25",
                                "Weight": 1
                            }
                        ],
                        "TargetGroupStickinessConfig": {
                            "Enabled": false
                        }
                    }
                }
            ],
            "IsDefault": true
        }
    ]
}
[ec2-user@ip-10-0-1-218 20210817-13:15:43]$
```

Screenshot of opening http://52.223.23.164/from quarantined IP address	 Not Secure | 52.223.23.164 **Welcome to this vulnerable webserver** Neat.	The screenshot confirms that when visiting the website from a quarantined IP address, the request is forwarded by the load balancer to WS2 honeypot webserver based on the new listener rule.
Screenshot of non-quarantined IP address opening http://52.223.23.164/	 Not Secure | 52.223.23.164 **Welcome to this awesome webserver** Neat.	The screenshot confirms that the load balancer forwards requests from non-offending IP addresses to WTG target group where WS1 is the registered target.

Screenshot of modified listener in the AWS console	 WebserverLB | HTTP:80 (2 rules) Rule limits for condition values, wildcards, and total rules. 1 arn...cadef IF ✓ Source IP is 87.80.156.133/32 THEN Forward to HoneypotTG: 1 (100%) Group-level stickiness: Off last HTTP 80: default action *This rule cannot be moved or deleted* IF ✓ Requests otherwise not routed THEN Forward to Webserver-target-group: 1 (Group-level stickiness: Off	The screenshot confirms that the Lambda function ELB_update successfully created the new rule.

Test ID 4.1

Test step	Test output	Test analysis

1. NMAP scan in normal mode	`nmap -sV 52.223.23.164`
	`Starting Nmap 7.91 ( https://nmap.org ) at 2021-08-17 20:26 BST`
	`Nmap scan report for a24ce6b897106d380.awsglobalaccelerator.com (52.223.23.164)`
	`Host is up (0.010s latency).`
	`Not shown: 999 filtered ports`
	`PORT   STATE SERVICE VERSION`
	`80/tcp open  http    Apache httpd 2.4.48 (())`
		NMAP scan started from regular IP address hitting WS1

2. NMAP scan in defence mode	`nmap -sV 52.223.23.164`
	`Nmap scan report for a24ce6b897106d380.awsglobalaccelerator.com (52.223.23.164)`
	`Host is up (0.014s latency).`
	`Not shown: 999 filtered ports`
	`PORT   STATE SERVICE VERSION`
	`80/tcp open  http    Apache httpd 2.4.33 (())`
		NMAP scan started from quarantined IP address hitting WS2

Test ID 4.2

Test step	Test output	Test analysis

1. NMAP scan in normal mode	`nmap -sV 52.223.23.164`
	`Nmap scan report for a24ce6b897106d380.awsglobalaccelerator.com (52.223.23.164)`
	`Host is up (0.012s latency).`
	`Not shown: 999 filtered ports`
	`PORT   STATE SERVICE VERSION`
	`80/tcp open  http    Apache httpd 2.4.48 (())`
		NMAP scan started from regular IP address hitting WS1

2. NMAP scan in defence mode	`nmap -sV 52.223.23.164`
	`Nmap scan report for a24ce6b897106d380.awsglobalaccelerator.com (52.223.23.164)`
	`Host is up (0.013s latency).`
	`Not shown: 999 filtered ports`
	`PORT   STATE SERVICE VERSION`
	`80/tcp open  http    Apache httpd 2.4.48 (())`
		NMAP scan started from quarantined IP address hitting WS2

Test ID 4.3

Test step	Test output	Test analysis
1. Start NMAP scan from non-quarantined IP address	`nmap -sV 52.223.23.164`	NMAP scan starts from regular IP address

2. At the same time as step 1 quarantine IP address by	`[ec2-user@ip-10-0-1-218 ~]$ aws lambda invoke --function-name ELB_update response.`
	`json`
	`{`
	`    "StatusCode": 200,`
	`    "ExecutedVersion": "$LATEST"`
		Status code 200 is successful execution of the Lambda function.

executing Lambda function ELB_update	`}`	

3.Capture NMAP scan results	`Nmap scan report for a24ce6b897106d380.awsglobalaccelerator.com (52.223.23.164)`
	`Host is up (0.010s latency).`
	`Not shown: 999 filtered ports`
	`PORT   STATE SERVICE VERSION`
	`80/tcp open  http    Apache httpd 2.4.33 (())`
		NMAP scan results show scan result hitting WS2

Test ID 5.1

Test step	Test output	Test analysis
1.Stop WS2 EC2 instance		

1.Execute start_ec2 Lambda function	`aws lambda invoke --function-name start_ec2 response.json`
	`{`
	`    "StatusCode": 200,`
	`    "ExecutedVersion": "$LATEST"`
	`}`
	`[ec2-user@ip-10-0-1-218 20210818-13:36:35]$`
		The Lambda function start_ec2 was executed at 13:36:35

2. Record timestamp when WS2 is accepting requests	`curl -Is http://35.71.153.62 | head -1`
	`HTTP/1.1 200 OK`
	`date`
	`Wed 18 Aug 2021 13:37:13`
		The website became available at 13:37:13. The time elapsed between executing the Lambda function start_ec2 until the website is available is 38 seconds.

3. Execute launch_ami Lambda function	`aws lambda invoke --function-name launch_ami response.json`
	`{`
	`    "StatusCode": 200,`
	`    "ExecutedVersion": "$LATEST"`
	`}`
	`[ec2-user@ip-10-0-1-218 20210818-14:31:35]$`
		Lambda function launch_ami was executed at 14:31:35

Execute HTG_update Lambda function	`aws lambda invoke --function-name HTG_update response.json`
	`{`
	`    "StatusCode": 200,`
	`    "ExecutedVersion": "$LATEST"`
	`}`
	`[ec2-user@ip-10-0-1-218 20210818-14:32:05]$`
		Lambda function HTG_update was executed at 14:32:05

Record timestamp when new EC2 webserver is accepting requests	`curl -Is http://35.71.153.62 | head -1`
	`HTTP/1.1 200 OK`
	`date`
	`Wed 18 Aug 2021 14:32:21`
		The website was available at 14:32:21. The time elapsed between executing the Lambda function launch_ami until the website is available in 46 seconds.