“Oppgave 1”

Task 1.

Explaining the requirements.

“The solution should ensure segregation of duty between platform and infrastructure administrators as well as secure infrastructure, and platform access.”

Segregation of duty in Amazon Web Services (AWS) involves dividing tasks and responsibilities among different individuals or roles to ensure that no single entity has excessive control over critical operations. This segregation is particularly important between platform and infrastructure tasks. For instance, individuals responsible for managing the underlying infrastructure, such as servers, may have different roles than those handling the platform, such as application deployment. This helps mitigate the risk of unauthorized access or misuse of privileges.

“The solution should enforce least privilege for all resources, S3 buckets, users etc.”

Ensuring least privilege is a security principle that dictates granting individuals or systems only the minimum level of access or permissions required to perform their designated tasks or functions. The goal is to restrict access rights for users or applications to the essentials necessary for their roles or responsibilities withing AWS. This provides an enhanced security posture and mitigates potential security risks. By adhering to this principle companies can minimize the potential impact of a security breach, data exposure or other malicious activities. AWS provides tools like Identity and Access Management (IAM), which allows granular control over user permissions.

“The solution should support both account and network isolation.”

AWS recommends using separate accounts for different purposes to achieve isolation. Each AWS account operates independently with its own set of resources, users, and permissions. This separation enhances security, resource management, and compliance by creating distinct boundaries between different projects, teams, or business units.

Network isolation is achieved through Virtual Private Clouds (VPCs) in AWS. VPCs allow users to create isolated network environments, define IP address spaces, and configure subnets. Security Groups and Network Access Control Lists (NACLs) provide control over inbound and outbound traffic within the VPC, enhancing security and segmentation.

“The solution should implement network security including defense in-depth.”

Defense in depth is a security strategy that involves implementing multiple layers of security controls and measures to protect a system or organization. The idea behind defense in depth is to create a series of barriers or defenses, each adding an additional layer of protection. This approach aims to provide redundancy and

resilience, making it more difficult for attackers to penetrate and compromise the system.

“The solution should ensure application security.”

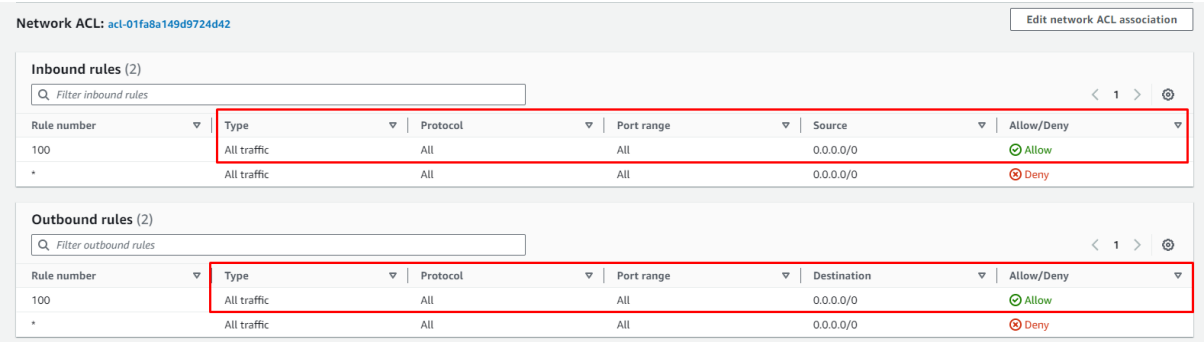
Securing applications in AWS involves implementing a combination of services and best practices. This includes securing APIs, using encryption for data in transit and at rest, implementing proper authentication and authorization mechanisms, and leveraging services like AWS Web Application Firewall (WAF) for protection against web application attacks.

(AWS, *Security best practices in IAM, 2023)*

(AWS, *Security Pillar, 2023)*

Analyzing the solution and determining if there are any security risks.

Both the Private subnet and the public subnet has the same ACL. This should be separated to allow further specification on what traffic shall be allowed to each respective subnet. The already configured ACL has security risks in the configuration. The current ACL allows traffic from every IP on All ports. This is a major security issue and should be closed off. For the public subnet take into consideration what ports will be needed for the application and regulate them accordingly. For a web application, consider only allowing HTTP on port 80 and HTTPS on port 443. The private subnet should only allow traffic from trusted sources, such as the internal network or specific subnets.

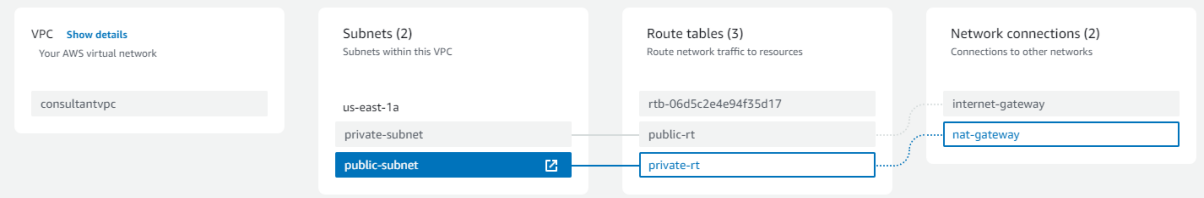


The public routing table and private routing table is switched around.

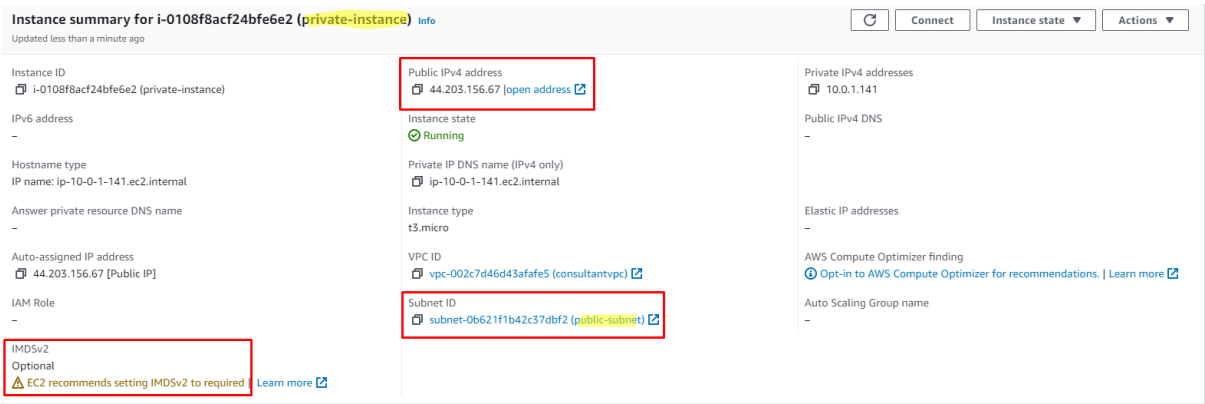
The public subnet should be linked to the public route table which in turn is connected to the internet-gateway. This allows the public subnet to communicate with the internet.

The private subnet should be linked to the private route table which in turn should only have a NAT gateway, allowing for the subnet to communicate to services

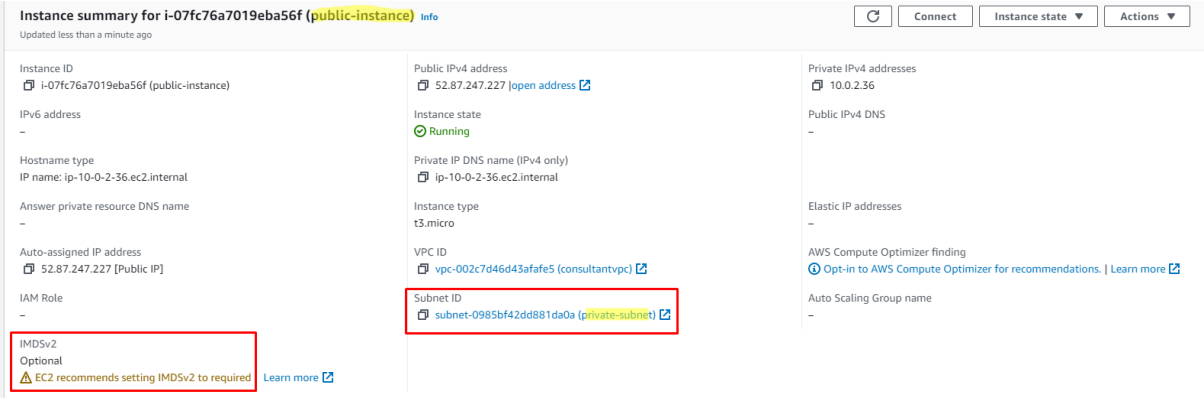
outside the VPC, but will not allow external communication back to the instance.



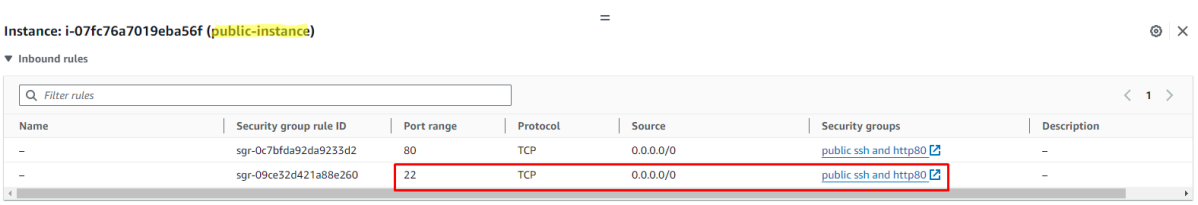
The private EC2 instance provides several security flaws. The IMDSv2 is set to optional, however best practice for security is to set this to require IMDSv2 metadata with a token requirement. The private EC2 instance is misplaced on the public-subnet, allowing for access from the internet to the private instance. The instance which should be private should also not have a public IPv4 address, only local.



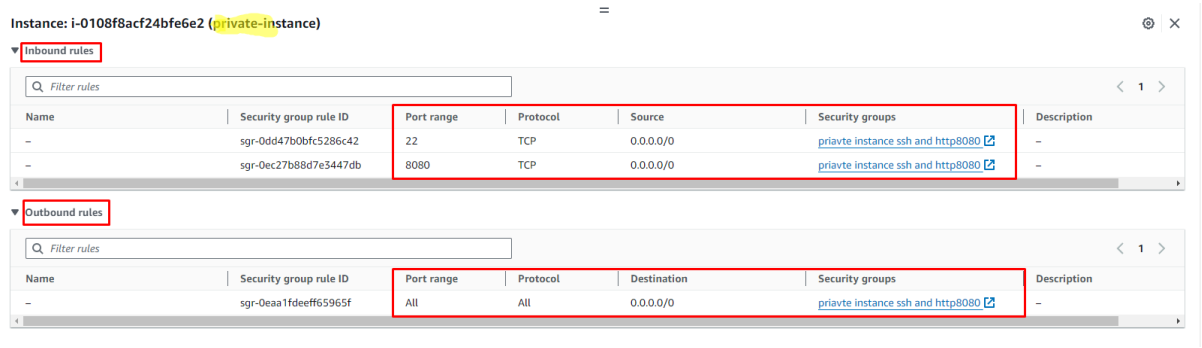
The public EC2 instance is also vulnerable. Here the IMDSv2 is also set to optional and should be forced as well. The instance also runs on the private subnet, causing the instance to act as private and not being able to receive communications from the internet.



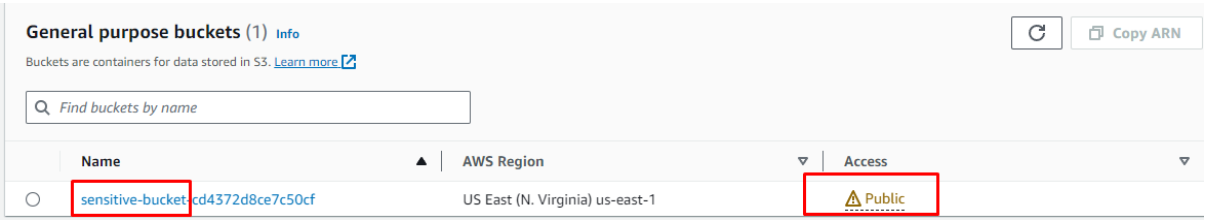
The public EC2 instance allows for all inbound traffic on port 80 (HTTP) and port 22 (SSH). SSH should be only specified to specific IP addresses if there is an administrative need or turned off entirely. Otherwise only allow for port 80 and port 443).



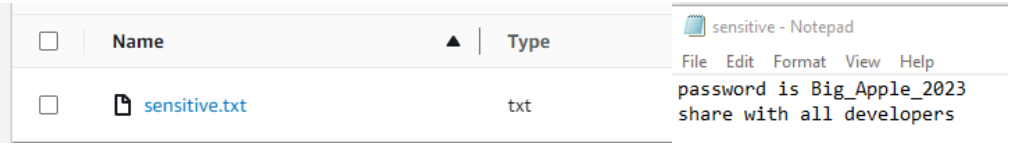
As for the private EC2 instance, this allows for all inbound traffic on port 22 and 8080. This is a big vulnerability as the access to the private instance should be very restricted. This private instance should only allow traffic from other instances withing the same security group or subnet, based on what is necessary for communication and only open ports as necessary to for these communications. As for the outbound traffic, the instance allows all IP addresses to access every port. This is also very insecure and should also be restricted to other security groups or subnets. Always follow the principle of least privilege in these cases and restrict all access to begin with and then allow necessary and inbound traffic based on what is required.



The S3-bucket should not be public access. This is a security flaw, exposing sensitive information to the internet and should be treated as highly sensitive. This bucket should be private and to further follow best practices, should be renamed to not contain identifying details about the sensitive content in the name.



The content of the bucket is sensitive information, containing a .txt file with credentials that is being shared between all developers. This does not follow security best practices and breaks with lest privilege and segregation of duty. Each developer should have their own credentials with restrictive access to strictly necessary tasks to enforce least privilege and help with monitoring of actions. Passwords should also never be stored in plain text.



Task 2.

Analyzing the consultants' suggestions.

1. *According to the consultant, the terraform script should be run using the access key ID and secret access key of AWS account root user to ensure easy deployment*

It is strongly discouraged to employ the access key ID and secret access key associated with the root user in AWS during the deployment of Infrastructure-as-Code (IaC) scripts using the Terraform tool. This caution is due to the root user's possession of expansive and unrestricted privileges, rendering the use of such credentials susceptible to security vulnerabilities. The credentials of the root user ought to be treated with the utmost sensitivity and refrain from routine usage.

Adhering to and aligning with Medicom's requirement for least privilege and account isolation, it is advisable to establish a distinct Identity and IAM role and user exclusively provided with the essential permissions requisite for the script's specific functionalities. This approach mitigates the danger of compromising the root user, preventing potential unauthorized entities from obtaining unrestricted access. Utilizing dedicated IAM users and roles enables enhanced continuous monitoring of the AWS solution by simplifying the tracking of actions performed by specific users.

To enhance the solution's adherence to security best practices, the creation of a dedicated IAM user tailored for deploying the Terraform script is recommended. This user should be provided with permissions strictly in line with the requirement of the task, refraining from overly permissive policies. Additionally, the secure storage of credentials, utilizing AWS CLI, is advised, while the practice of hardcoding credentials within Terraform files is to be avoided to fortify the overall security posture.

2. *“The private key of the SSH key pairs should be stored on the public instance to enable easy SSH access to the private instance.”*

The imperative confidentiality of the SSH key pair's private key mandates that it be safeguarded and exclusively accessible by its rightful owner. The inadvertent exposure of the private key to the public sphere represents a substantial security vulnerability, counteracting the fundamental objectives of key-based authentication. This lapse in security introduces the perilous prospect that any user gaining access to the public instance may exploit the divulged private key to infiltrate other instances linked to the key pair. Such unauthorized access poses a severe threat to the integrity of the system and the potential compromise of sensitive data.

To adhere to the best practices of key management, keys should be encrypted and managed securely. It is imperative to ensure the secure storage, sharing and revocation of keys. Regularly rotating keys should also be practiced, as this is a proactive measure to mitigate the inherent risks associated with compromised private keys. The adoption of the AWS Key Management Service (KMS) is recommended as a facilitative means to effectively manage keys within the framework of a secure cryptographic environment.

3. *“Three tier architecture is not necessary because both the internal and private tiers are not accessible from the internet, therefore the script only implements 2-tier architecture with public and private subnets in one account.”*

The adoption of a two-tier architecture may be suitable for modest applications with specific requirements and needs. Nevertheless, in the context of Medicoms, such an architectural configuration is deemed inadvisable. As an alternative, a three-tier

architecture is recommended, introducing an intermediary tier between the web server and the database server. This additional layer provides a comprehensive defense-in-depth strategy, thereby expanding network security and ensuring segregation—a requirement by Medicoms.

To enhance the solution, the implementation of a three-tier architecture is vital. This entails the creation of additional IAM users and roles tailored to administer each newly introduced segment of the network. This approach aligns with best practices, assuring optimal security measures and substantively meeting Medicom's requirements for the proposed solution.

4. *“Instance metadata service should be set to optional.”*

The AWS Instance Metadata Service (IMDSv2) is an improved version of the metadata service and should always be set to required to enhance security when it comes to accessing instance metadata. The IMSDv2 is requires the use of session tokens when requesting metadata, thus helping protect against various vulnerabilities and attacks, such as server-side request forgery and theft of tokens. The service also prevents token theft, making it challenging for token theft to be exploited, resulting in better control of actions of the users in the system and preventing unauthorized access to users. Some applications in AWS also rely on metadata to function properly. It is therefore highly recommended to make use of the IMSDv2 and set it to “required”.

(AWS, *Use IMDSv2, 2023*)

(AWS, *Security best practices in IAM, 2023)*

(AWS, *Security Pillar, 2023)*

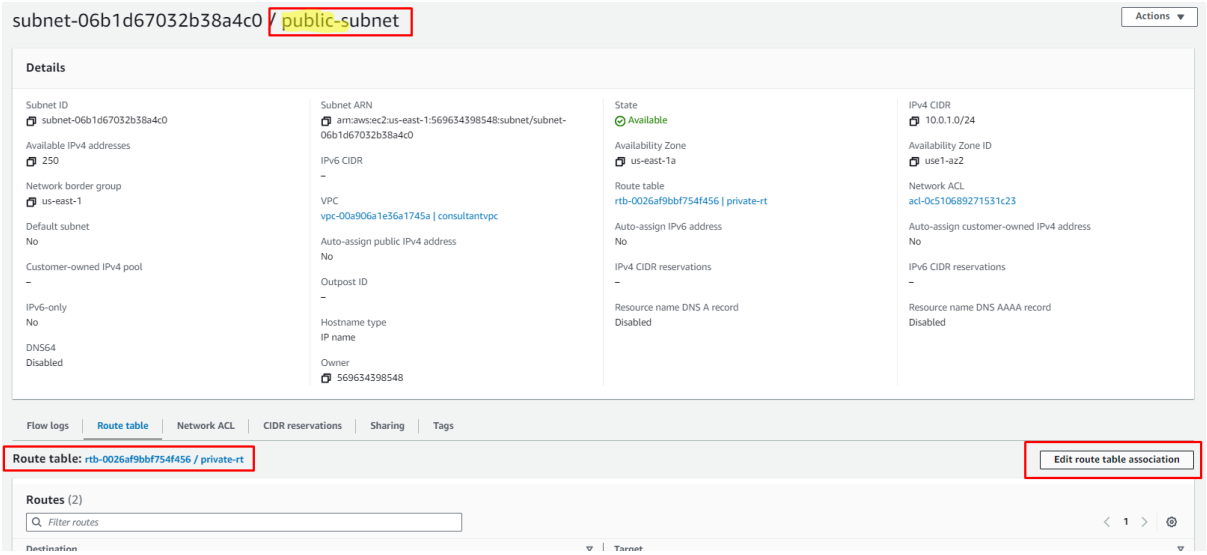
Task 3.

*Disclaimer
The following critical security issues are not configured in order of severity or chosen based on how severe they are in relation to one another or other issues in the AWS solution provided.

1. Private and Public subnet has same ACL rules
2. Overly liberal ACL rules for subnet traffic / Lax ACL rules for network traffic
3. Change the IMDSv2 to required
4. Overly liberal Security Groups for EC2 instances
5. Public S3 Bucket with sensitive information.

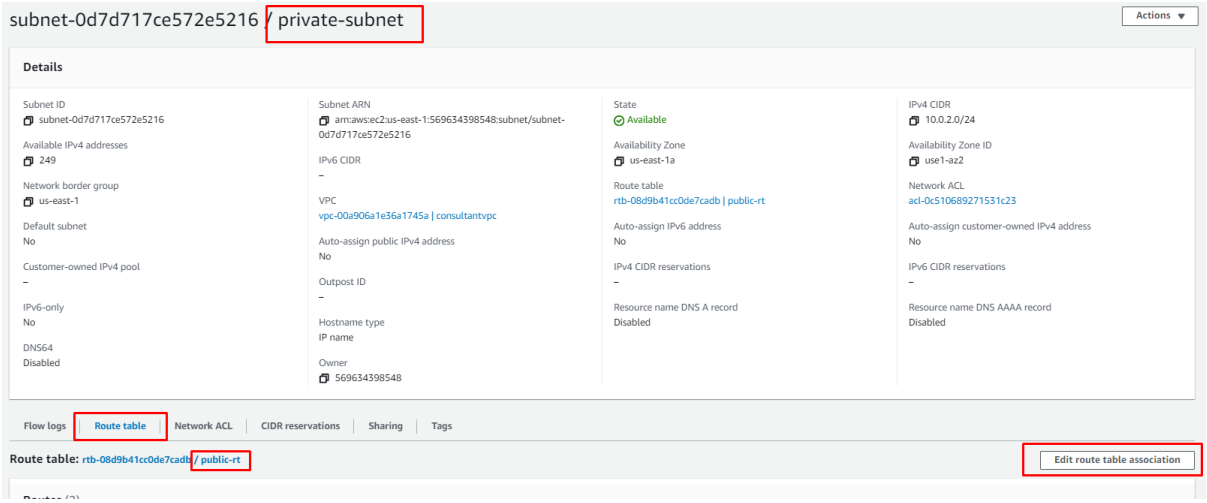
1. **Private and Public subnet has same ACL rules.**

First off, I start with navigating to the public subnet and click the “Route table” pane. As you can see the public subnet is attached to the private-rt (bottom left). I then click the “Edit route table association” button, bottom right.

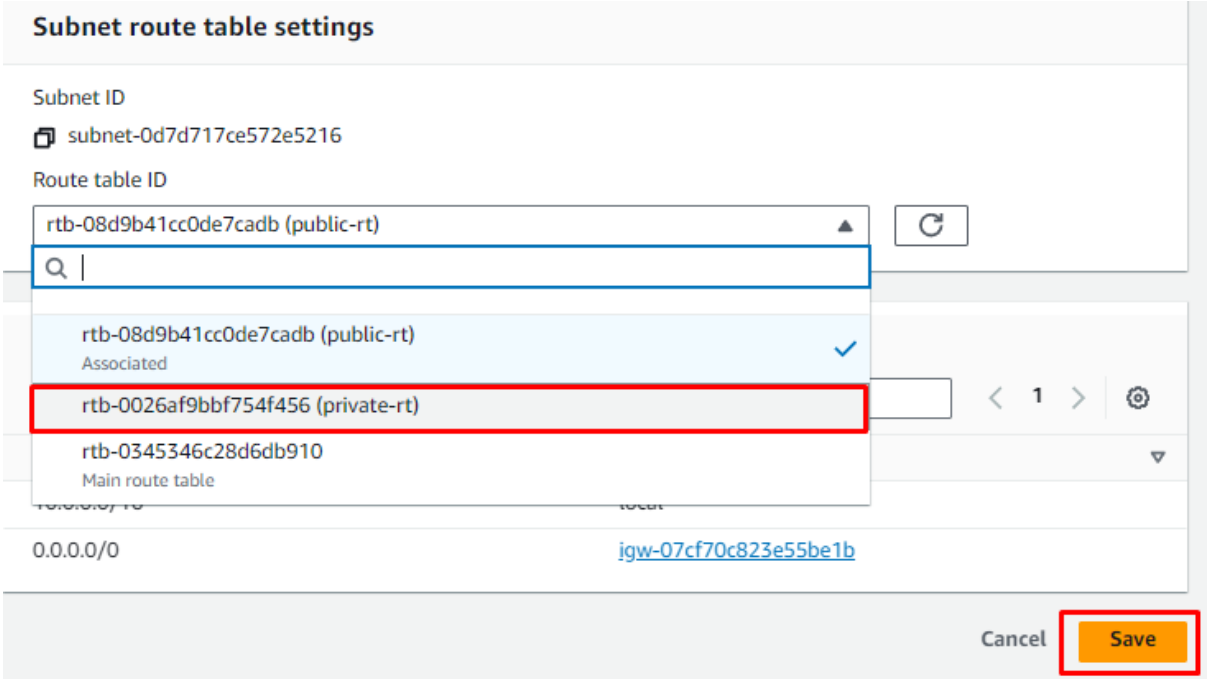


Here we simply click the dropdown table, and we can swap to the public-rt for the subnet ending in 4c0. Which we know is the public subnet (See previous photo). Afterwards, click “Save”.

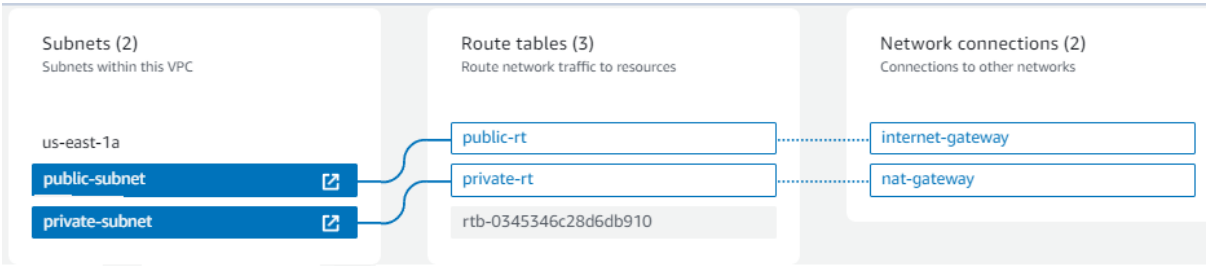
We simply do the same again, but for the private subnet. Navigate to the subnet, click “Route table” pane and “Edit route table association”



Next, we click the drop-down menu, and change the route table to the private-rt, so that it matches the subnet. Afterwards, click “Save”.

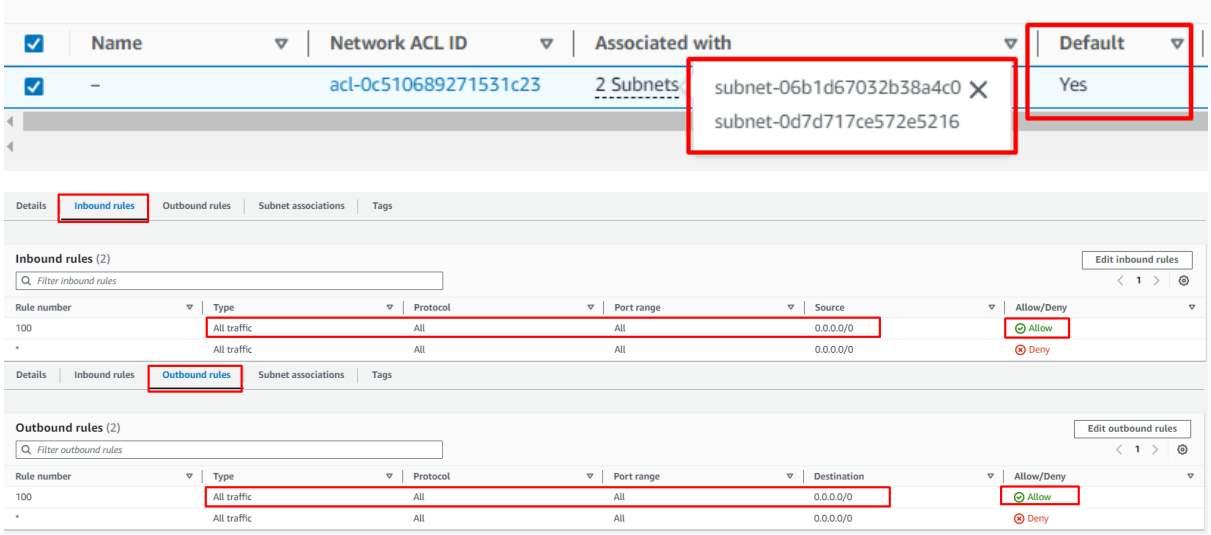


Now the subnets are correctly associated with the respective route tables and then again the correct network connections. Like this.

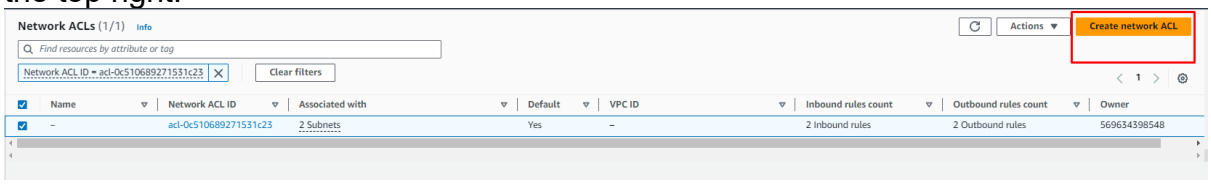


2. Overly liberal ACL rules for subnet traffic / Lax ACL rules for network traffic.

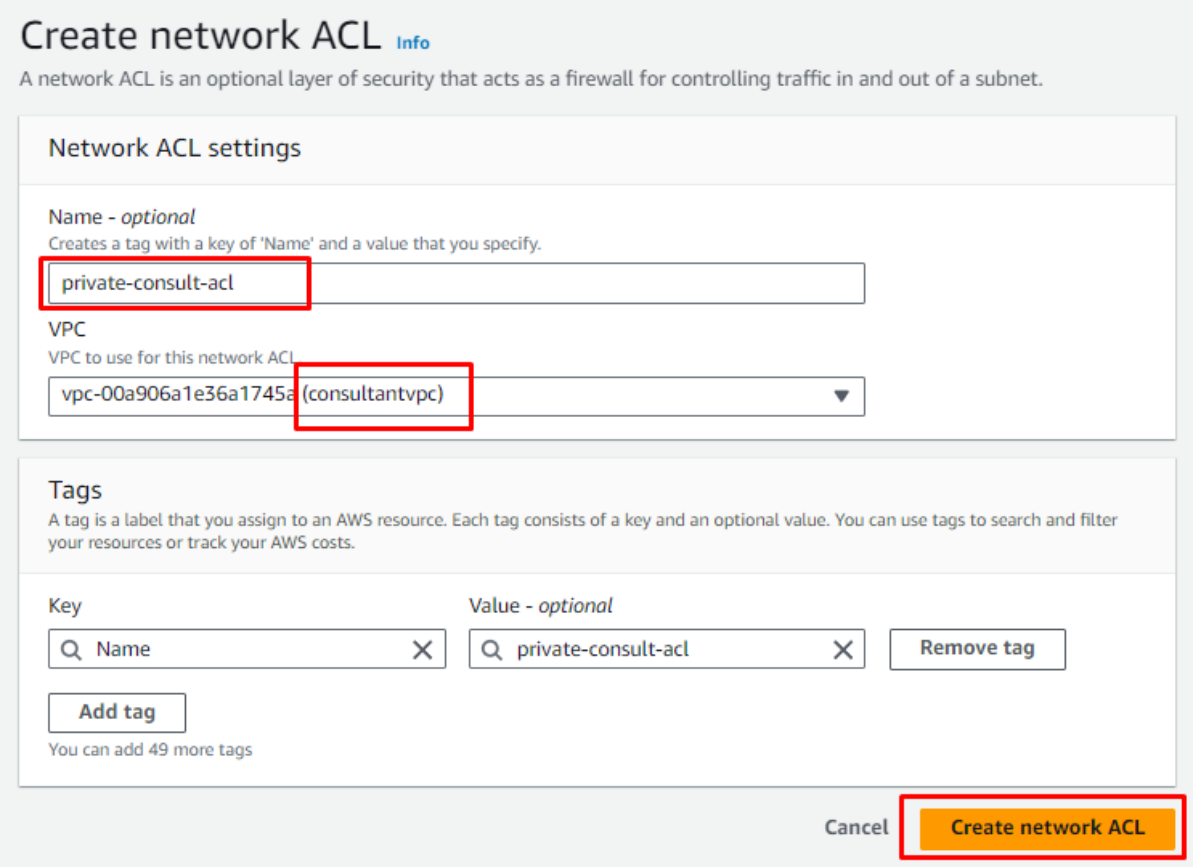
The provided solution only has one default network ACL which is configured to regulate both the subnets with very lax traffic rules. This breaks the best practice for network security, defense in depth, network isolation and secure infrastructure. The rules of the ACL also allow for unrestricted traffic, both to and from the network, resulting in a major security risk for the company. To fix this we start with creating a new ACL.



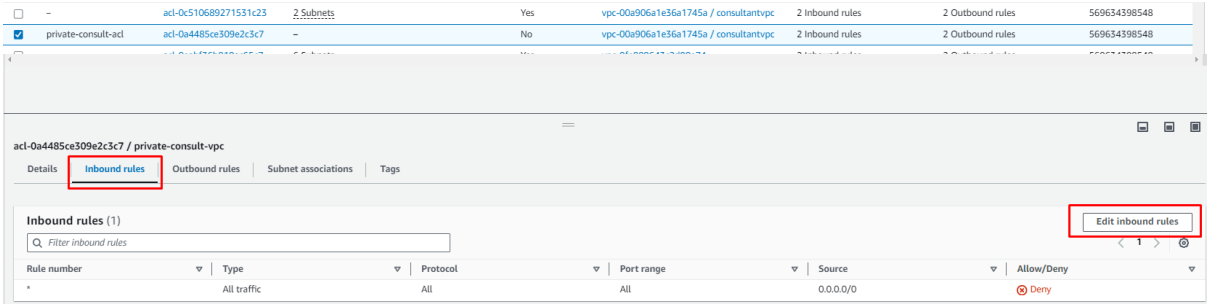
To fix this, we start with creating a new ACL by clicking the “Create network ACL” in the top right.



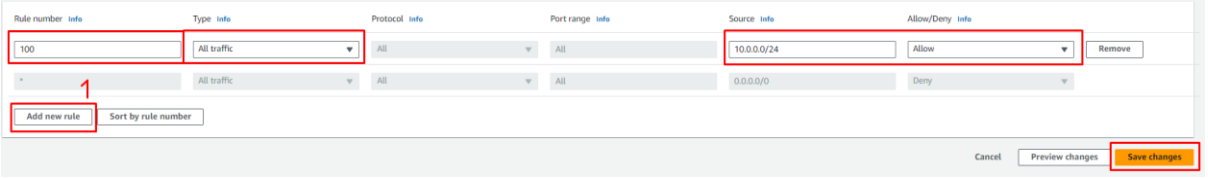
We then give it a descriptive name, like “private-consult-acl” to explicitly tell any user that this is for the private network. Make sure to attack it to the correct VPC, consultantvpc and click create.



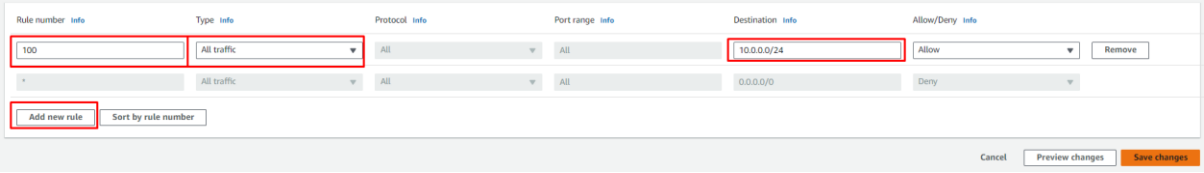
Now that we have our new ACL for the private subnet, we can select it and start editing the rules. Start with “Edit inbound rules”



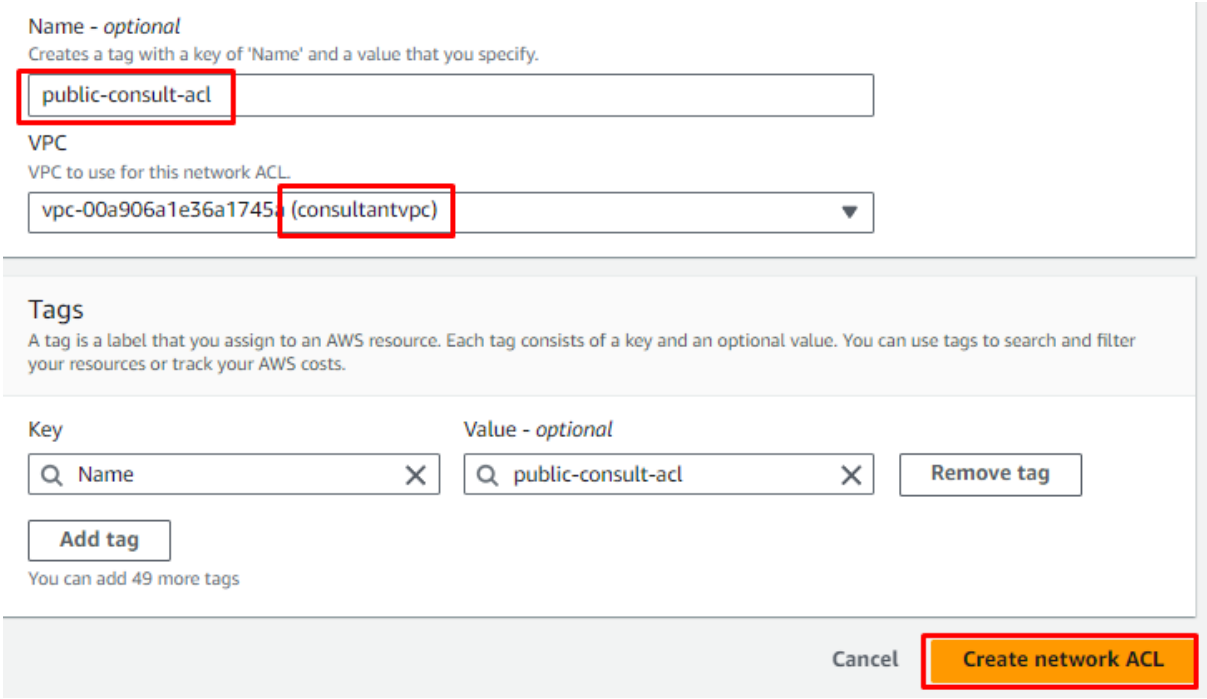
Once in the inbound rules, click the “Add new rule” and give it rule number 100, this makes it our priority rule. This rule will allow for all traffic on all ports, but only within our private network. We specify this by providing the local address “10.0.0.0/24”. Hit “Save changes” afterwards.



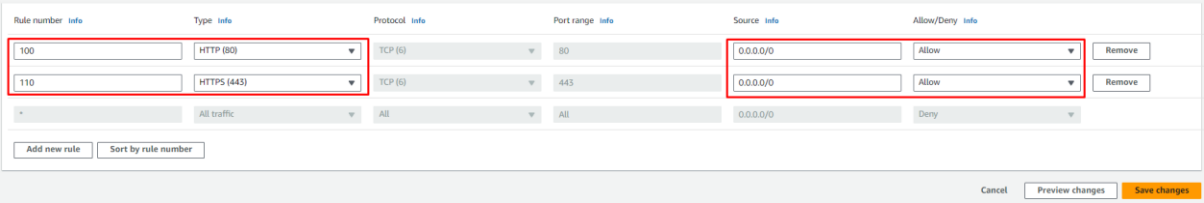
We do the same for the outbound traffic rules. We create a new rule with number 100 and allow all traffic to the local IP of “10.0.0.0/24”. The reason for this, is that while we could allow for outbound traffic to the internet, as of now we have no information for the necessity of this and to follow the principle of best practice, we would rather start with a more restrictive solution and open as needed.



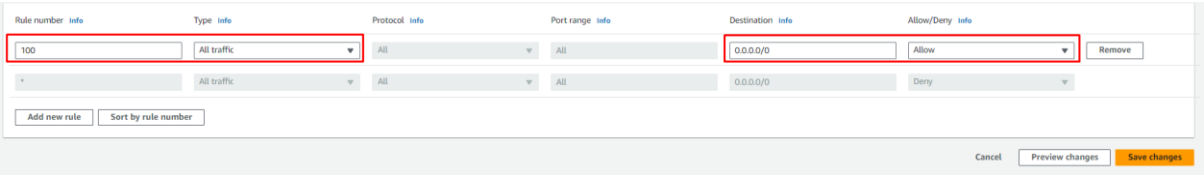
Now that the ACL for the private subnet is created, we repeat the same process for creating an ACL for the public subnet. We name it respectively “public-consult-acl” and attach it to the consultantvpc.



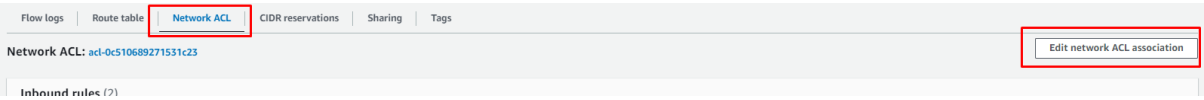
Select the new “public-consult-acl” in the overview of Network ACLs and chose “Inbound rules” and click “Edit inbound rules”. For this solution, I will assume Medicoms needs to allow for traffic to web severs and needs to allow for this traffic. We therefore add a new rule, giving it number 100, allowing for normal HTTP traffic over port 80 for every IP, with source “0.0.0.0/0”. This will result in the ACL allowing all traffic over HTTP and HTTPS, and if any traffic is coming from any other port the universal “*” rule will deny the traffic. Our inbound rules loos like this before we save the changes.



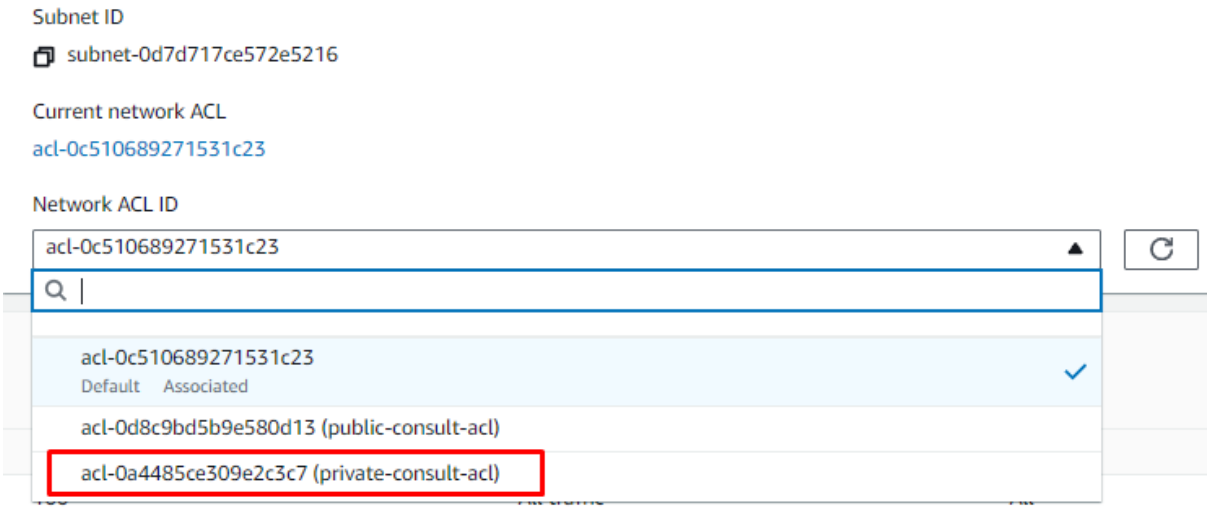
Now, navigate to the “Outbound rules” pane, and click “Edit outbound rules”, with the “public-consult-acl” still selected.
We add a new rule, give it number 100, and allow for “all traffic”. This allows for communication with the internet with no specific restrictions. Like so:



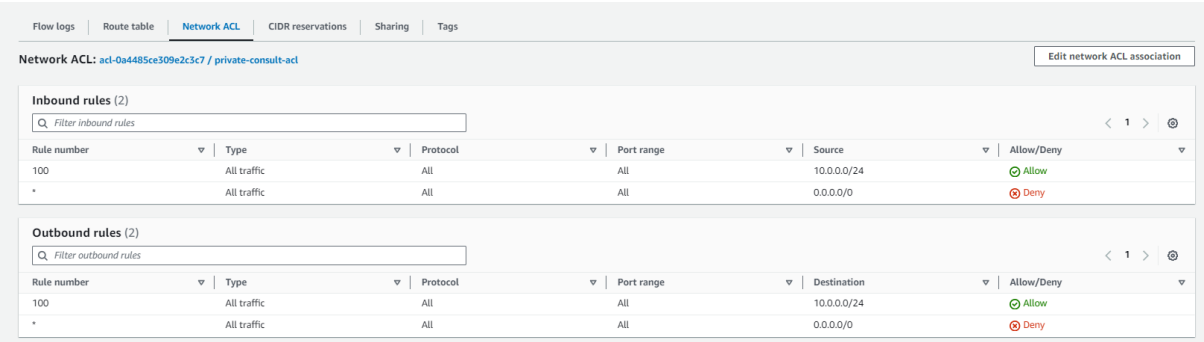
Now we need to apply our new ACL rules to the correct subnets for them to work. We navigate to on of our subnets, I will start with the “private-subnet”. I then click on “Network ACL” and then “Edit network ACL association”.



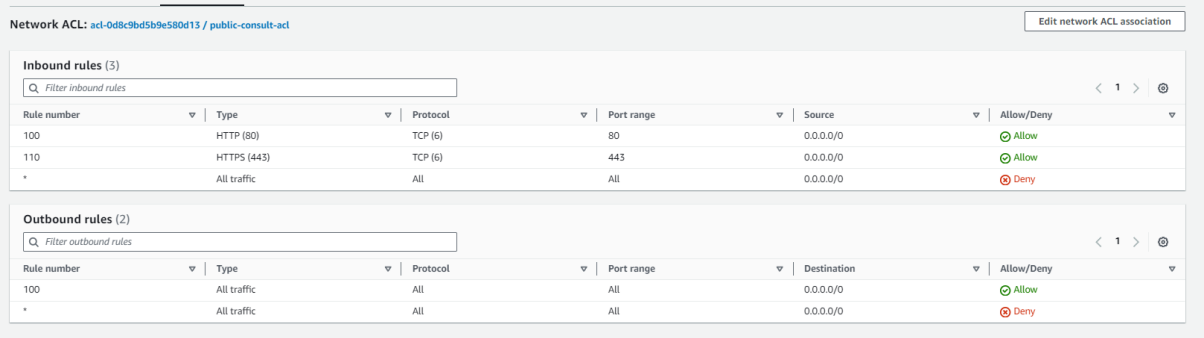
Here we click on the drop-down menu, and select the correct ACL, in this first case the “private-consult-acl”. Repeat this for the “public-subnet” and chose the respective “public-consult-acl”.



The private-subnet ACL should now look like this:

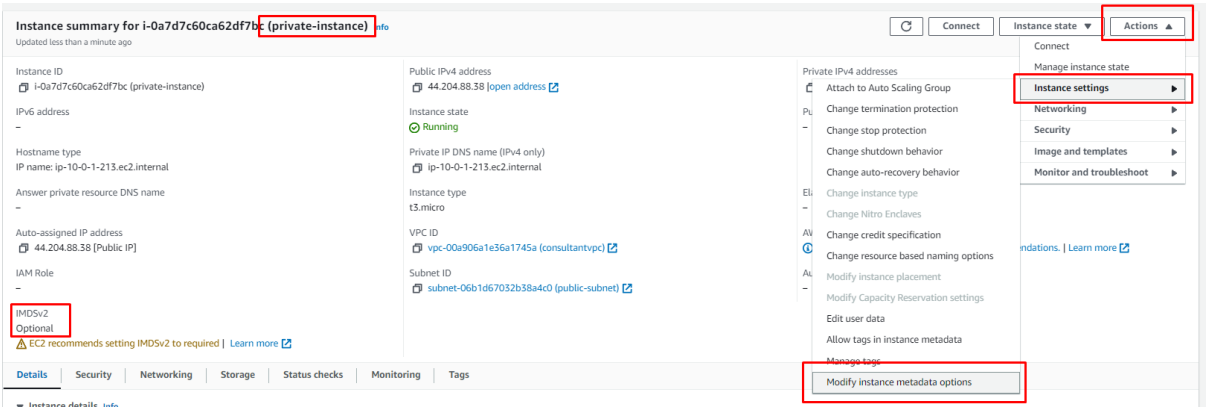


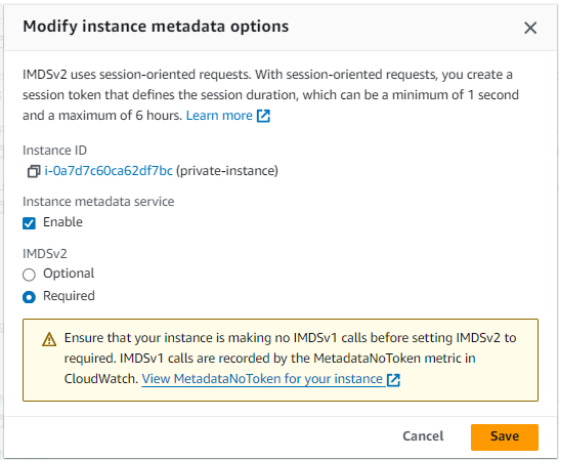
And the public-subnet ACL should look like this:



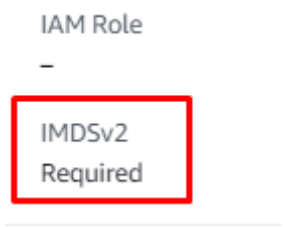
3. Change the IMDSv2 to required

Both the private and public EC2 instance has IMDSv2 set to optional. It is recommended to always set this to required. We can easily do this, by clicking on the “Action” drop-down menu, selecting “Instance settings” and finally “Modify instance metadata options”. Select the “Required” option and press “Save”.



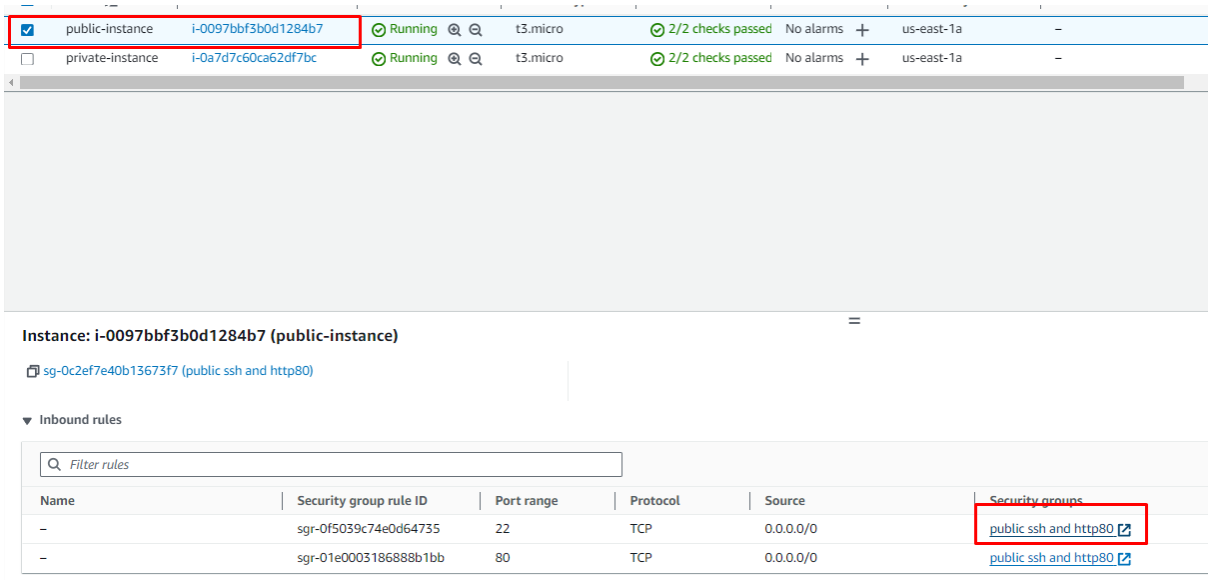


Repeat these same steps for both the public and private EC2 instance and they should both display IMDSv2 as “Required”, like this:



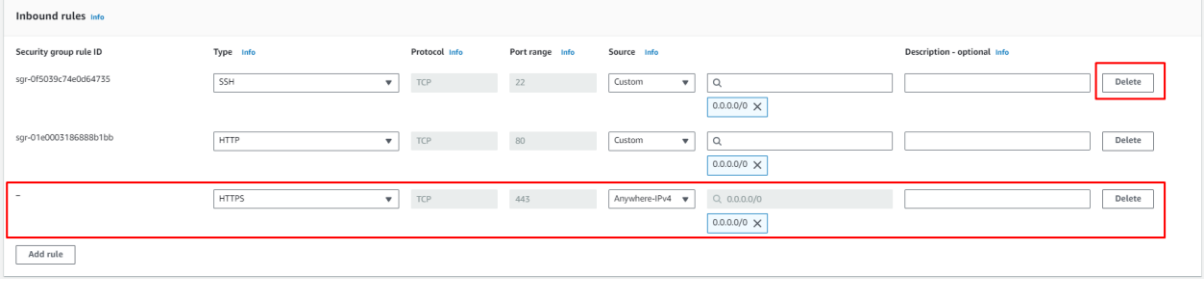
4. Overly liberal Security Groups for EC2 instances

Both the public and private instance of EC2 use Security Groups with overly liberal rules allowing for excessive access to the instances. This is a severe security issue giving unauthorized access to the instances. To fix this we will implement more restrictive rules to the security groups. Select an instance and then click the security group attached. Here shown with public instance and security group.

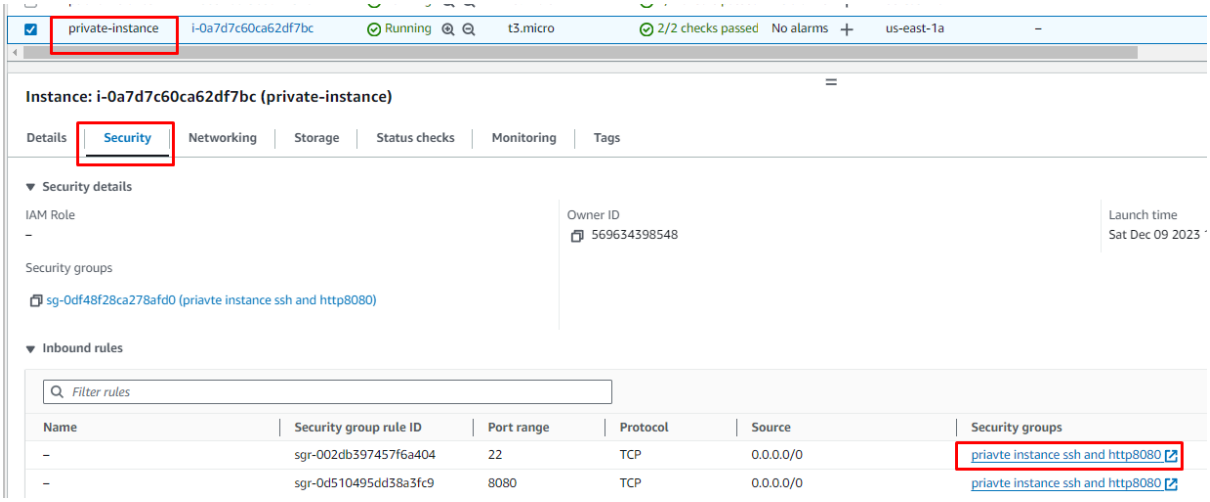


Select the "Security Group, “Inbound rules” and then “Edit inbound rules”. While SSH could be used for remote administration of the system, this should not be enabled for all IPs on the internet. If this is a necessity, specifically assign IP addresses to this rule. In our case, we do not have any information about any user needing the SSH option, and we will delete the rule for now.

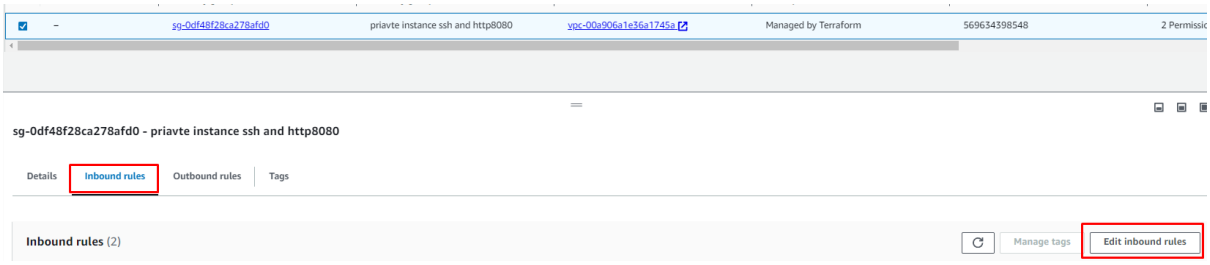
The rule for HTTP over port 80, to the internet is fine as is, and need no configuration. However, we can add another rule allowing HTTPS over port 443 as well. I leave the outbound rules as is.



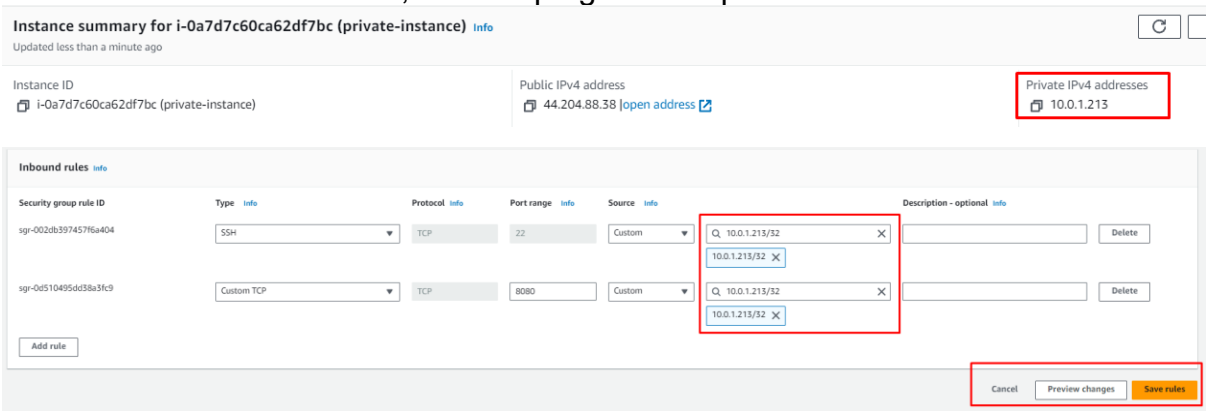
Now under the EC2 instances overview, select the private-instance, and then “Security” and select the group under “Security groups”



Now we navigate to “Inbound rules” and click “Edit inbound rules”.

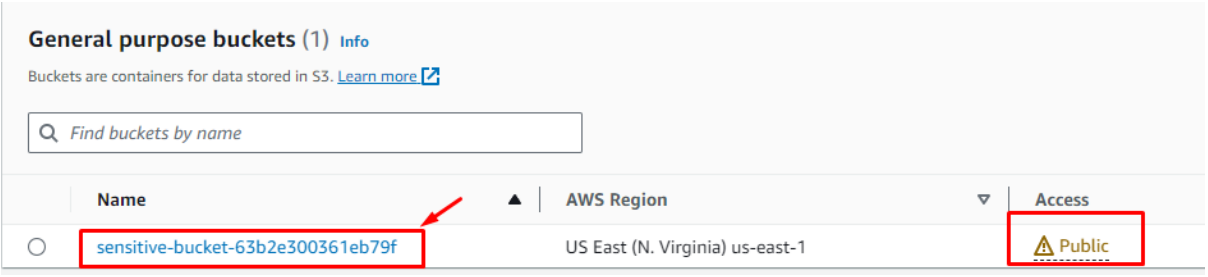


The security group already allows for SSH and TCP on port 8080 from all IP addresses, which should not be the case. The private EC2 instance should not allow for traffic beyond the security group. We therefore leave the rules, but change them to the local IPv4 CIDR block, found top right of the private EC2-instance.

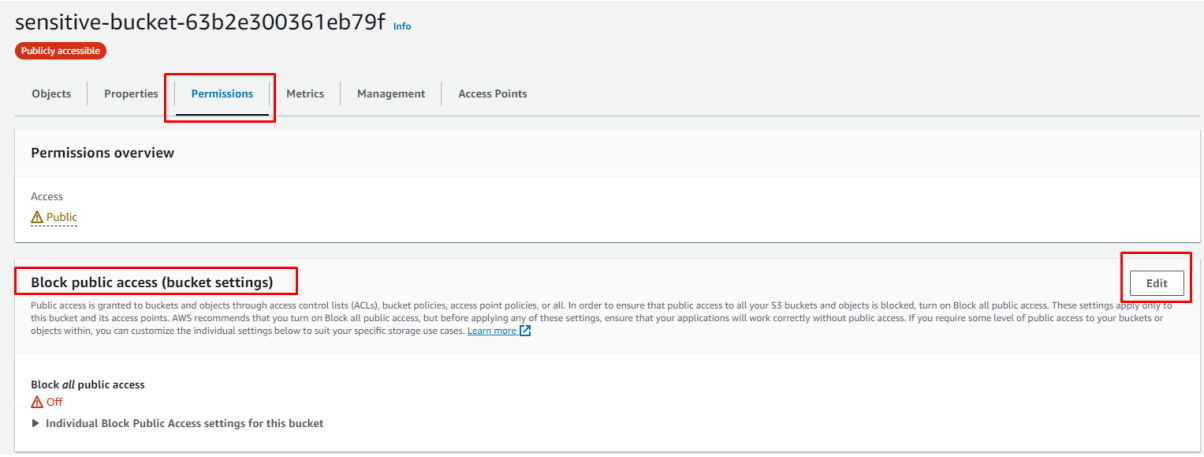


5. Public S3 Bucket with sensitive information.

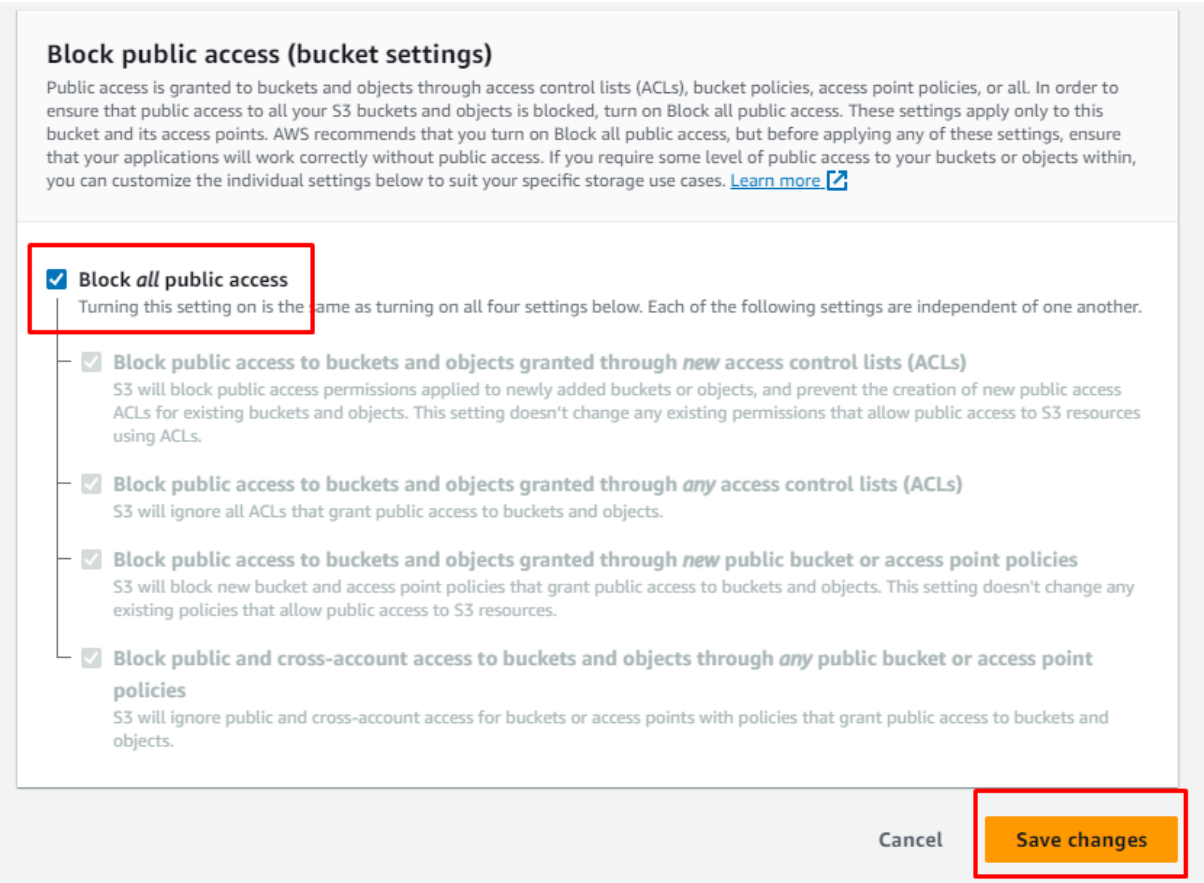
The S3 bucket created in AWS, with the name “sensitive-bucket-…” is set to public. Considering this is a bucket containing sensitive information should be treated as such and restricted to be private. To do this, we navigate to the S3 section and click on the bucket.



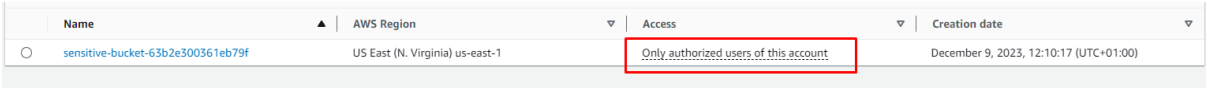
Once inside we can click on the “Permissions” pane, and navigate down to the “Block all access” section, to click on “Edit”.



We toggle the option to “Block *all* public access” and save. We write confirm when prompted.



Our bucket is not only accessible by “authorized users of this account”. If anyone wants to gain access, they need specifically configured IAM users. This helps with segregation of duty, least privilege and account isolation.



Further security faults that can be improved are, to name some: Private EC2 instance has public IP, Private EC2 instance is on public subnet and vice-versa**.**

“Oppgave 2”

Task 1.

This statement represents a distribution of security responsibilities between cloud service consumers and providers. Specifically, the customer assumes accountability for the protection and administration of their data and applications within the cloud environment. This encompasses the configuration of access controls, encryption protocols, and the implementation of secure networking practices to fortify the security posture of virtual machines, applications, and associated datasets.

In contrast, the cloud service provider is entrusted with ensuring the overall security of the cloud infrastructure itself, irrespective of individual user practices. This responsibility extends to the safeguarding of physical data centers, fortification of network architecture, and the implementation of comprehensive security frameworks at the broader cloud platform level. The provider's purview encompasses defense against external threats, with security protocols typically instituted at the infrastructure level.

(AWS, *Shared responsibility, 2023)*

Task 2.

NACLs function as stateless firewalls within the AWS infrastructure. These entities serve to regulate the traffic to and from a network within one or more subnets. Operating within a hierarchical framework, these firewalls assign priority to rules based on their designated rule numbers. Notably, only the initial rule aligning with the characteristics of a given network packet is executed, rendering subsequent rules obsolete.

In this specific scenario, the administrator of an Elastic Compute Cloud (EC2) instance encounters a connectivity impediment with the Google DNS server. This issue arises due to the configuration of Rule 100 within the NACL, which explicitly denies all traffic originating from the source IP address range of 0.0.0.0/0. Consequently, as every IP address corresponds to this range, the rule indiscriminately blocks all traffic attempts, thereby precluding the EC2 instance administrator from reaching the Google DNS server. Despite the Google DNS server

having its own rule, the predetermined priority structure of rules renders it ineffectual in this context.

(AWS, *Security best practices in IAM, 2023)*

(AWS, *Security Pillar, 2023)*

Task 3.

In AWS, inbound traffic refers to the data directed towards an AWS resource from external sources, including user-generated requests or data transmissions. Conversely, outbound traffic refers to the data exiting the resource and being transmitted to external destinations, which may involve requests originating from users or systems external to the AWS environment.

The regulation of inbound and outbound traffic is managed through the establishment of rules within Security Groups and NACLs. Both mechanisms afford the capability to govern and limit traffic based on specified IP address ranges. Notably, Security Groups operate at the instance level, whereas NACLs function at a subdomain level. This distinction allows for a nuanced approach in delineating communication permissions and restrictions for AWS resources.

Task 4.

Effective policy implementation follows a principle of least permission. The efficacy of permissions is dependent upon the presence of coherent policies. Consequently, practical access is constrained to the minimum granted level. Illustratively, if an IAM Policy grants S3 Full Access while the Permission Boundary permits only S3.GetObject, the ultimate access level is restricted to the more limited permission, specifically, access solely to S3.GetObject.

In the context of the tasks assigned, the Service Control Policy is applicable to both IAM and Permission Boundary. However, a noteworthy absence of a shared policy between IAM and Permission Boundary invokes the least permissive policy, effectively rendering it nonexistent. Consequently, the resulting effective policy denies any requests due to the absence of common permissions between IAM and Permission Boundary.

(AWS, *Security best practices in IAM, 2023)*

(AWS, *Security Pillar, 2023)*

Task 5.

If a properly configured private AWS instance encounters challenges in fetching updates from the internet, the issue may be attributed to constraints on outbound traffic. Given that security groups exercise authority over both inbound and outbound data flows, overly restrictive outbound rules can impede the instance's ability to establish communication with external repositories or internet-based update servers.

To address this issue, one should examine the outbound rules within the security group associated with the private instance. Ensure the defined rules permit the necessary outbound traffic and adjust them if needed.

Task 6.

Continuous monitoring in cybersecurity characterized by the persistent and systematic observation and analysis of the security posture of an information system. This approach is instrumental in the identification and response to security events, ensuring an organization's ability to proactively address potential threats in a dynamic environment. Unlike conventional one-time scans, continuous monitoring establishes an ongoing and real-time framework for the collection, analysis, and responsive management of data.

AWS provides several continuous monitoring services, notably Amazon CloudTrail, Amazon CloudWatch, and AWS GuardDuty.

Amazon CloudTrail facilitates the observation of AWS cloud deployments by generating detailed logs of API calls. This functionality aids in the identification of users and accounts associated with API calls, offering essential details such as source IP addresses and timestamps for accountability.

Amazon CloudWatch, characterized by its adaptability and scalability, provides real-time data collection from AWS resources. This capability enables users to monitor and gain insights into the operational health and performance of deployed resources through customizable dashboards and alarms.

AWS GuardDuty, a specialized threat detection service, employs machine learning for continuous surveillance of AWS accounts and workloads. It identifies instances of malicious activity and unauthorized behavior, delivering security findings for informed remediation strategies.

Task 7.

To quickly fix the SQL injection vulnerability, implementation of strategic measures to enhance security within an AWS environment involves the use of outbound rules to mitigate data exfiltration risks. Employing an AWS WAF is crucial, wherein the creation of rules becomes essential in the mitigation of SQL injection attacks. This application firewall possesses the capability to examine and impede malicious traffic proactively, intercepting potential threats before they reach the targeted application. One effective approach involves the initial denial of all traffic, gradually permitting access to designated ports as required. Additionally, restricting all outbound traffic altogether, where no need for communication beyond local access exits, adds an additional layer of security.

Furthermore, optimizing security against SQL injection threats within AWS encompasses the utilization of a forward proxy. This proxy assumes the role of an intermediary between client requests and the web applications. The forward proxy serves as a formidable defense mechanism by meticulous filtration and inspection of incoming data. This process enables the identification and potential sanitization of

data before its transmission, thereby fortifying the defense against SQL injection vulnerabilities.

Task 8.

The provided log is of the type: Flow Logs. The data can be indexed using the following template:

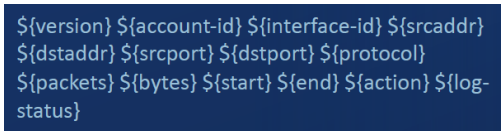


figure 1.

After indexing we have the following overview of the log.

VPC Flow Log Fields	Associated values in the VPC flow log
version	2
account-id	12345678910
interface	eni-12345b8ca123456789
source ip-address	172.31.16.139
destination ip-address	172.31.16.21
source port	20641
destination port	22
protocol (IANA number)	6
packets	20
bytes	4249
start time unix	1418530010
end time unix	1418530070
log-status	REJECT OK

figure 2.

In this log, the VPC flow log version used is 2.
The ID for the AWS account where the log was created is 12345678910.
The ENI interface-ID from which the log was captured is eni-12345b8ca123456789.
The IP-address of the source for the log is 172.31.16.139 from the port 20641, while the destination IP-address is 172.31.16.21 and used port 22.
The IANA protocol number for the traffic is 6.
The log shows 20 packets transferred, containing 4249 bytes of data.
The log start time was 1418530010 in UNIX time, meaning Sun Dec 14, 2014, 05:06:50 GMT+1. The log ended 1418530070 UNIX time, meaning that it ended Sun Dec 14, 2014, 05:07:50 GMT+1, just 1 minute later.
Finally, the log-status shows REJECT OK, telling us the traffic was rejected.

Figure 1. is taken from Lecture 10-Logging and monitoring-v1.pdf, slide 21.
Figure 2. Is based on the table in slide 22.

Sources:

AWS. (2023, December 6th) *Security Pillar - AWS Well-Architected Framework* *https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/welcome.html*

AWS. *Security best practices in IAM* retrieved December 9th, 2023, from *https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html*

AWS. *Security foundations* retrieved December 9th, 2023, from *https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/security.html*

AWS. *Shared responsibility* retrieved December 9th, 2023, from https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/shared-responsibility.html

AWS. *Use IMDSv2* retrieved December 9th, 2023 from https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html

Ebenezer, P. (2023). *Logging and Monitoring-v1* [Slide presentation]. Canvas. https://kristiania.instructure.com/courses/11234/files/1250041?module_item_id=431437