HBO-ICT | Hogeschool van Amsterdam

Cloud Security Posture Management

A Research Driven Approach to improve AWS security by detecting and classifying cloud misconfigurations.

Auteur
Robin van Dijk

Mentor
Richard de Koning

Company
Niagaros

Departement
hbo-ict cs

Date
9-Dec-25

Version
0.3

Creating Tomorrow

Managementsamenvatting

Table of contents

1. Introduction

Context

Organizations increasingly rely on hyperscale cloud platforms, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud to deploy, scale, and manage their applications. While these platforms offer flexibility and operational efficiency, they also introduce complex security challenges. Misconfigurations are one of the most frequent causes of cloud security incidents, often resulting from human error or insufficient visibility into the environment.

As this environment continues to grow, manually verifying configurations and identifying security risks becomes increasingly challenging. Common issues such as overly permissive IAM roles, improperly configured S3 buckets, or missing logging settings can lead to data exposure or unauthorized access if not detected in time.

Niagaros

Niagaros is a security company that focuses on helping organizations build, maintain, and secure their cloud environments mainly on Amazon Web Services (AWS). The company works on setting up cloud infrastructures, improving existing environments, and solving technical issues related to networking. They also support clients with cost optimization and general cloud management.

In addition to engineering work, Niagaros carries out security-related activities. This includes reviewing access policies, checking resource configurations, and performing penetration tests on both cloud setups and applications. Their goal is to identify gaps that could lead to security problems, such as exposed data, weak permissions, or missing monitoring.

Although Niagaros supports multiple clients, the company works especially closely with Domits, which is their main partner. For Domits, Niagaros handles a large part of the AWS environment, such as deploying new resources, maintaining existing infrastructure, and helping improve their overall security posture. Because of this long-term involvement, Niagaros has a clear understanding of the challenges Domits faces particularly when it comes to keeping track of configuration issues in a growing cloud environment.

This close collaboration is also the reason Niagaros is exploring the development of its own CSPM prototype. The aim is to give Domits a simple way to detect misconfigurations, understand risks, and maintain a secure cloud environment without relying on manual checks. Niagaros wants to achieve this while also creating a product that can scale and be used for future partners.

Problem statement

In recent years, Niagaros have seen their clients struggle with maintaining a secure cloud posture as their environments grow. Misconfigurations, such as overly permissive access policies, insufficient logging, or unprotected resources are not clear for their clients. This is why Niagaros decided to start developing its own Cloud Security Posture Management (CSPM) platform.

Currently, Domits does not have an automated way to assess the security posture of its AWS environment. To address this gap, Niagaros wants to explore how an initial CSPM prototype could be developed that automatically analyzes configurations, detects misconfigurations, and classifies the associated risks. Before such a prototype can be designed, insight is needed into the existing AWS setup, the relevant security standards, and the functional and technical requirements for the solution.

Central research question and sub-questions

This leads to the main research question of this graduation project:

“How can Niagaros develop a CSPM prototype that analyzes AWS configurations, detects security misconfigurations, and classifies the associated risks?”

To answer this question, the study is structured around four sub-questions:

SQ1. What does the current AWS environment of Domits look like, and which configurations or risks are present?
SQ2. What security standards, best practices, and CSPM methodologies exist for assessing AWS security configurations?
SQ3. What functional and technical requirements are needed to build an MVP CSPM tool that can analyze resources, settings and rate risks from AWS?
SQ4. How can the CSPM prototype be designed and implemented to reliably detect, classify, and present security risks to Domits?

Reading guide

Chapter 2 provides an overview of the context surrounding this research, introducing the problem, the motivation for the study, and the research objectives. Chapter 3 explains the research methodology, outlining the approach used to gather information, analyze the current AWS environment, and derive requirements for the CSPM prototype.

Chapter 4 presents the literature review, covering existing security standards, AWS best practices, and methodologies used by modern Cloud Security Posture Management tools. This chapter forms the theoretical foundation for understanding how misconfigurations occur and how automated security analysis can be structured.

Chapter 5 examines the current AWS environment of Domits in detail. It identifies the existing configurations, potential risks, and recurring security issues. This analysis highlights the practical problems that the CSPM prototype must address. Based on these findings, Chapter 6 defines the functional and technical requirements for the prototype.

Chapter 7 describes the design and implementation of the CSPM proof-of-concept, including its architecture, data collection process, risk classification logic, and reporting mechanisms. Chapter 8 evaluates the performance and reliability of the prototype, discussing its strengths, limitations, and opportunities for improvement.

Finally, Chapter 9 answers the central research question by synthesizing the results from all previous chapters. It also provides recommendations for further development of the CSPM solution and suggestions for future research.

Readers primarily interested in the practical results of this project may focus on Chapters 5, 7, and 8, while those interested in the methodological or theoretical aspects may refer to Chapters 3 and 4.

2. Theoretical Framework & methodology

Approach

This research combines qualitative and quantitative methods to understand Domits' AWS environment and develop the CSPM prototype. Qualitative methods provide context about processes and decision-making, while quantitative and technical methods produce objective measurements of security risks. The approach includes document analysis, stakeholder consultation, configuration scanning, benchmarking, and prototype testing.

Methods used in this section:

- Document analysis — used for SQ1
- Stakeholder interviews — used for SQ1 and SQ3
- Technical configuration analysis — used for SQ1
- Compliance checks — used for SQ1
- Misconfiguration testing — used for SQ4
- Prototype iteration — used for SQ4

Literature research

The literature research focuses on cloud security standards, CSPM methodologies, and AWS best practices. This includes reviewing the CIS AWS Benchmark, NIST 800-53, ISO 27001, AWS Well-Architected Framework, and academic papers on cloud misconfigurations and CSPM models. The goal is to identify how misconfigurations should be detected, classified, and evaluated.

Methods used in this section:

- Literature review — used for SQ2 and SQ3
- Comparative study of CSPM tools — used for SQ2
- Benchmarking of CSPM features — used for SQ3

Applied Research

Applied research was performed directly in the AWS environment of Domits. This includes scanning regions, evaluating service configurations, validating compliance, testing detection logic, and assessing prototype performance. Simulated misconfigurations and controlled experiments were used to determine how effectively the prototype can detect and classify risks.

Methods used in this section:

- Technical AWS configuration analysis (AWS CLI, region scans) — used for SQ1
- Compliance assessment (CIS / AWS best practices) — used for SQ2
- Simulated misconfiguration testing — used for SQ4
- Prototype development and evaluation — used for SQ4

3. Current AWS environment

Services

Amplify

The AWS Amplify environment for Domits currently hosts multiple active branches for the web application. The main production branch (main) is deployed to https://domits.com, and the acceptance branch (acceptance) is deployed to https://acceptance.domits.com.
Both branches are continuously deployed from the backend develop environment, ensuring that updates to the application are automatically reflected on the live sites.

Each branch is linked to a separate deployment environment, with the latest commits automatically triggering deployments. The most recent deployments occurred within the last few hours, indicating an actively maintained and frequently updated setup.

From a research perspective, this configuration highlights key aspects of the current AWS environment: the use of continuous integration and deployment pipelines, live staging and production environments, and publicly accessible domains. These elements are important to note for potential risks, such as misconfigured permissions, exposure of sensitive endpoints, or unreviewed code changes being automatically deployed, which could impact security if not properly managed.

S3

In the examined AWS environment, there are multiple S3 buckets with varying functions, regions, and creation dates. Examples include accommodation, accommodationphotos, maciesensitivebucket, as well as various deployment and log buckets such as amplify-domits-production-150926-deployment and aws-cloudtrail-logs-115462458880-3afc7874. Each bucket is located in a specific AWS region, such as Stockholm (eu-north-1) or Ireland (eu-west-1), and contains data ranging from media files and web application deployments to sensitive financial or personal information.
It is important that the initial security audit focuses on high-risk buckets, including:

- Buckets with sensitive or personal data – for example, maciesensitivebucket or domits-host-payout-invoices. Access to these must be strictly limited, data should always be encrypted, and features like MFA Delete or versioning should be applied.
- Deployment and log buckets – such as amplify-domits-production-150926-deployment and aws-cloudtrail-logs-115462458880-3afc7874. These are less sensitive in terms of personal data, but they must not be publicly accessible, and encryption and logging should be properly configured.
- General buckets – such as media files (accommodationphotos) or temporary storage. Although these are less critical, public access settings and lifecycle policies should be reviewed to ensure compliance.

The first step in an S3 security audit is to check public access, bucket policies, ACLs, encryption, logging, versioning, and lifecycle rules. By following this systematic approach, risks such as unauthorized access, data leaks, or compliance issues can be identified and addressed early.

Cloudwatch

The current AWS CloudWatch environment for Domits consists of 238 log groups, covering a wide range of services such as API Gateway, Lambda functions, AppSync APIs, and other backend applications. Each log group is configured as Standard, with anomaly detection and deletion protection currently turned off, and retention set to never expire by default.
Notable observations include:

1. Extensive Lambda coverage: Most Lambda functions have their own dedicated log groups, ranging from business-critical functions like AddUserToGuestGroup and FetchGuestPayments to web application functions such as PhotoUploadFunction and StripeWebHook. This indicates that the environment logs all Lambda invocations, which is good for traceability.

2. Retention settings: The fact that all log groups are set to never expire can be problematic. While this ensures full historical logging, it may lead to excessive storage costs and difficulty in managing older logs. From a security perspective, it also means that any sensitive information logged will persist indefinitely.

3. Lack of anomaly detection and deletion protection: All log groups have anomaly detection turned off and deletion protection not enabled. This represents a potential risk because abnormal patterns or accidental deletion of logs might go unnoticed.

4. Public or sensitive exposure: No explicit sensitive data counts are reported, but with 238 log groups, it's likely that some contain sensitive information (e.g., user IDs, payment data, email addresses). Without proper monitoring and access control, these logs could be inadvertently exposed to unauthorized users.

5. API Gateway and AppSync logging: Log groups such as /aws/apigateway/ranhr5t8qe/production and /aws/appsync/apis/773tt5aqtbhfdgew7heayjptwy indicate that requests and operations to API endpoints are logged. This is essential for auditing API usage and identifying potential misconfigurations or security events.

The CloudWatch setup provides comprehensive logging for all key components in the AWS environment, but there are potential areas for improvement regarding retention policies, deletion protection, and anomaly detection. From a research perspective, these log groups are critical for understanding system behavior, tracking user interactions, and identifying security risks across the Domits AWS environment.

Cognito

AWS Cognito is used within the Domits AWS environment to manage user authentication, authorization, and identity federation. The setup consists of five Cognito User Pools and three Cognito Identity Pools, primarily created and managed through AWS Amplify for different environments such as development and production.

User Pools:
The following user pools are currently active:

- Amplify backend manager user pool, used internally by Amplify to manage backend environments.
- Domits development user pool, which is actively updated and used for non-production authentication flows.
- Domits production user pool, responsible for authenticating end users of the live application.
- Two additional standalone user pools, likely created for testing or experimental purposes.

User pools function as directories for end users and support login mechanisms such as email/password and potentially federated identity providers. The separation between development and production user pools indicates a structured environment that reduces the risk of test users or configurations affecting production systems.

Identity Pools:
Three identity pools are present:

- An Amplify backend manager identity pool.
- A Domits development identity pool.
- A Domits production identity pool.

Identity pools are used to provide temporary AWS credentials to authenticated users, enabling controlled access to AWS resources such as S3 or API Gateway. These pools are typically linked to user pools and rely on IAM roles to define permissions.

Lambda

AWS Lambda is widely used in the Domits AWS environment and forms the core of the backend. Many Lambda functions handle application logic such as bookings, messaging, payments, onboarding, and integrations with external services like Stripe and Google APIs. The functions are mainly triggered through API Gateway, S3 events, Cognito triggers, scheduled jobs, and WebSocket connections, showing a serverless and event-driven setup.

Most functions use Node.js runtimes, with several versions active at the same time, including Node.js 16.x up to 24.x. Some functions have not been updated for over a year, which may mean they are running on older dependencies. Using multiple runtime versions increases maintenance effort and makes it harder to apply consistent security updates.

The function names show a clear split between production, development, and test workloads. However, test or unused functions still increase the overall attack surface if they remain deployed. Each Lambda function runs with its own IAM role, so incorrect or overly broad permissions could lead to unauthorized access to services such as S3 or DynamoDB.

All Lambda functions log to Amazon CloudWatch, but log retention is set to never expire. This can result in long-term storage of sensitive data and makes it harder to monitor relevant security events over time.

For this research, Lambda is a key component of the AWS environment. While the setup supports scalability and fast development, differences in runtimes, aging functions, and logging settings may introduce security and management risks.

IAM

AWS IAM is heavily used in the Domits AWS environment to manage access for people, applications, and AWS services. The setup shows a clear separation between human users, service roles, and automated workloads.

IAM user groups are used to organize access based on roles and responsibilities, such as administrators, developers, billing, security, and read-only users. Groups like *Domits-Admin*, *Domits-Dev-Team*, *Domits-Security-Hub*, and *Domits-ReadOnly* indicate that access is mostly managed at group level instead of assigning permissions directly to users, which is a good practice. Some groups are service-specific, for example for SES, CloudFormation, or Secrets Manager, showing that IAM is also used to control access to individual AWS services.

There are many IAM users, mostly named after team members, which suggests that direct IAM users are still actively used instead of relying only on federated access. Several users have console access and active access keys, while some users or keys appear inactive or unused for long periods. This can increase risk if unused credentials are not cleaned up regularly.

IAM roles form a large part of the environment. Hundreds of roles exist, mainly used by AWS services such as Lambda, AppSync, EventBridge, EC2, and Amplify. These roles allow services to access other

AWS resources without using long-term credentials. The high number of roles reflects the serverless and event-driven architecture, where each function or service often has its own role.

The account also contains a large number of IAM policies, both AWS-managed and customer-managed. Managed policies are used for common access patterns, while custom policies are created for specific application needs, such as allowing Lambda functions to access DynamoDB or S3. This shows flexibility, but also increases complexity and makes oversight more challenging.

Security controls

Misconfigurations and Risks

Sub-Conclusion

4. Security standards

Cloud security standards

CIS AWS

ISO 27001

NIST

GDPR

Analysis of Standards

Sub conclusion

5. Functional and Technical requirements

Requirement gathering

User needs

Functional requirements

Technical requirements

Sub-Conclusion

6. CSPM prototype

Objectives

System Architecture

Implementation

Evaluation

Sub-Conclusion

7. Conclusion and Recommendation

Answers to the Sub-questions

Answer to the Main research question

Reflection of the prototype

Recommendation

Future implementations

8. References

9. Appendix

Appendix A: [Titel van de bijlage]

[Klik hier als u tekst wilt invoeren. Maak gebruik van de stijlen op de tab START voor het vormgeven van de tekst.]