



Privacy Sandbox Aggregation Service and Coordinator

Google
April 2, 2024

©2023 – NCC Group

Prepared by NCC Group Security Services, Inc. for Google LLC. Portions of this document and the templates used in its production are the property of NCC Group and cannot be copied (in full or in part) without NCC Group's permission.

While precautions have been taken in the preparation of this document, NCC Group the publisher, and the author(s) assume no responsibility for errors, omissions, or for damages resulting from the use of the information contained herein. Use of NCC Group's services does not guarantee the security of a system, or that computer intrusions will not occur.

Prepared By

Elena Bakos Lang
Giacomo Pope
Giovanni De Ferrari
Huy Nguyen
Lydia Yao
Thomas Pornin
Tyler Colgan
Viktor Gazdag

Prepared For

Google

1 Executive Summary

Synopsis

During the winter of 2022, NCC Group conducted an in-depth security review of the Aggregation Service, part of Google's [Privacy Sandbox](#) initiative. Google describes Privacy Sandbox as follows:

The Privacy Sandbox initiative aims to create technologies that both protect people's privacy online and give companies and developers tools to build thriving digital businesses. The Privacy Sandbox reduces cross-site and cross-app tracking while helping to keep online content and services free for all. One of the proposed solutions within the initiative is the [Aggregation Service](#). The goal of this service is to allow ad tech to generate [summary reports](#), which include aggregated measurement data on user's behavior collected by other Privacy Sandbox APIs; these APIs allow ad techs to collect aggregatable reports from clients. The aggregation service decrypts and combines the collected data from the aggregatable reports, adds noise, and returns a summary report. This service runs in a trusted execution environment (TEE), which is deployed on a cloud service that supports necessary security measures to protect this data. This approach is designed to provide a balance between protecting user privacy and meeting the needs of the advertising industry.

In spring 2023, NCC Group completed a retest on a series of fixes proposed by Google, and found that they effectively addressed all findings documented in this report.

Scope

NCC Group's evaluation included:

- **Web Services Assessment:** Dynamic testing and code review of the final design and deployment of the Privacy Sandbox Aggregation Service from the perspective of an external attacker.
- **Architecture Design Review:** Review of the final design of the Privacy Sandbox Aggregation Service.
- **Cryptographic Design and Implementation Review:** Comprehensive review of the cryptography implementation for the Aggregation Service and split key features.
- **Holistic Attacker-Modeled Pentest:** Holistic review of the final design and implementation of the Privacy Sandbox Aggregation Service from the perspective of a malicious ad tech firm.

Limitations

All testing materials needed for the engagement were provided prior to the start of testing. No testing impediments were experienced during the engagement.

Key Findings

Architecture Design Review

NCC Group did not identify any flaws in the design of Privacy Sandbox Aggregation Service. It appears to satisfy industry best practices and provide strong protections for the confidentiality and integrity of data collected from end users. The following principles from the service's design documentation were broadly respected:

- Uphold The Privacy Sandbox design goals for privacy protections in Attribution Reporting. Specifically, we intend to provide appropriate infrastructure for noise addition and aggregation, aligned with the long-term goal of differential privacy.



-
- Prevent inappropriate access to raw aggregatable reports or other intermediate data through technical enforcement.
 - Allow ad techs to retain control over the data they've collected and access noisy aggregated data without sharing their data with any third party.
 - Support flexible, scalable, and extensible aggregation strategies and on-demand access to the aggregation infrastructure, so that ad techs can choose when and how often to generate summary reports.
 - Provide open and transparent implementations for any infrastructure outside of the client.

Holistic Attacker-Modeled Pentest

There was no significant issue found that could allow an ad tech or any malicious party to gain access to any complete keys or higher privileges. There are some improvements suggested for the environment when an ad tech firm implements the demo environment by running the Terraform scripts. These include:

- Disable IMDSv1 to prevent attackers from gaining access tokens via SSRF attacks or querying IMDS.
- Update the enclave image for minimize attack surface.

Cryptography Design and Implementation Review

The overall design of the cryptography components within the Privacy Sandbox Aggregation Service was found to be suitable for the stated goals. The cryptographic algorithms chosen have sufficient security level and characteristics (e.g. all symmetric encryption is authenticated).

Strategic Recommendations

Include a patching policy timeline and process to address enclave image related vulnerabilities to prevent any vulnerabilities that would allow an attacker either to gain access to the system and control its runtime.

Ensure that the implementation and documentation match closely. Additionally, document the requirements for externally chosen cryptographic primitives, such as encryption keys chosen by the external coordinator, to ensure adequate security of the overall system.

Client Response

For build, base container images and instance images (AMI, GCE Machine Images) Google will proactively update those before every release¹ to stay up-to-date with security and vulnerability patches. In case of CRITICAL vulnerabilities in the enclave (and patches available for those) we will provide out-of-release cycle patching with patch releases and notify ad techs to update the version they are running to the newest patch version.

Both improvements recommended by NCC Group in the Holisitc Attacker-Modeled Pentest were found in the beta version of the Aggregation Service, and were addressed before General Availability.

1. <https://goo.gle/ps-aggregation-service-release-lifecycle>



2 Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors.

Title	Status	ID	Risk
Missing Public Key Integrity Check	Fixed	97Y	High
EC2 Instance Metadata Service Version 1 In Use	Fixed	7DQ	Low
Lack of Overwrite Controls in S3	Risk Accepted	9HH	Low
Lack of VM Image Hardening	Fixed	JPU	Low
Docker Image with Scan Findings	Fixed	R6C	Low
Container Image with Scan Findings	Fixed	TBC	Low
Lambda Function Without Code Signing	Risk Accepted	A44	Info
IAM Role Assigned with Excessive Permission	Risk Accepted	HWM	Info
Dynamo DB Alerting Not Enabled	Fixed	HH4	Info



3 Dashboard

Target Data

Name	Privacy Sandbox Aggregation Service
Type	Web service assessment, architecture review, host review, cryptography review
Platforms	Java, AWS
Environment	Test




Engagement Data

Type	Web services, host review, architecture review, cryptography & implementation review
Method	Code-Assisted
Dates	2022-12-05 to 2023-01-23
Consultants	7
Level of Effort	82 person-days






Targets

Privacy Sandbox Aggregation Service	Release version v1.0.1
	Beta version v0.8.0
Privacy Sandbox Coordinator	Release version 1.1.0
	Beta version 0.51.11

Finding Breakdown










Critical issues	0	
High issues	1	
Medium issues	0	
Low issues	5	
Informational issues	3	
Total issues	9	





Category Breakdown

Access Controls	2	
Auditing and Logging	1	
Configuration	4	
Cryptography	1	
Data Validation	1	



Component Breakdown

API	1	
Aggregatable Report Accounting Service	1	
Architecture Design Review	1	
Holistic Attacker Modeled Pentest	5	    
Secure Control Plane	1	

 Critical  High  Medium  Low  Informational



4 Architecture Review Methodology

The following document describes the methodology adopted by NCC Group to understand the Privacy Sandbox Aggregation Service architecture and design choices.

Google provided a number of documents covering design, specifications, security, and deployment scenarios for Privacy Sandbox Aggregation Service, which were reviewed by NCC Group.

Additionally, where possible, interviews were held with Google personnel in order to understand Privacy Sandbox Aggregation Service functionality, components, assets, and security controls.

During the aforementioned calls, questions about diagrams were raised and the expected outcomes and corner-cases discussed.

To support the question and answer process, a questionnaire was shared with Google in order to answer to a set of specific questions related to Privacy Sandbox Aggregation Service components.

After completing the initial interview and discovery phase, the system's security model was analyzed to review security assumptions and identify any gaps between them and those actually provided by the system. This allowed NCC Group to uncover possible weaknesses in the existing design as a result of misunderstood assumptions; missing security guarantees in the underlying networks, applications, or operating systems; and insufficient specifications.

Potential weaknesses that could appear due to changing assumptions or deployment scenarios were identified and documented.

Additionally, NCC Group reported a set of recommendations that could increase the security of the system, such as compensating controls that can provide defense in depth.

The complete analysis, observations and findings were provided in this report, supported by the creation of an architecture diagram for Privacy Sandbox Aggregation Service.



5 Architecture Review

This section evaluates the security decisions made by Google in designing the Privacy Sandbox Aggregation Service. NCC Group first identified design patterns which are common among secure networks and environments, then compared the design of the Privacy Sandbox Aggregation Service against best practices in each of those areas.

The Privacy Sandbox Aggregation Service was designed by Google to provide advertising services providers (ad techs) with an innovative, secure, and privacy-aware infrastructure that would allow them to collect users' individual aggregatable reports from browsers and devices, summarize them, and generate reports to be used for advertising purposes.

In order to deploy the Privacy Sandbox Aggregation Service, three agents were required, all independent from each other:

- One ad tech that took care of processing the aggregatable reports
- Two independent entities (Coordinator 1 and Coordinator 2) that took care of key management and aggregatable report accounting

Each one of the agents had to set up an AWS account that had to be configured according to the documentation that Google would provide them.

The aggregatable report data was processed within a Trusted Execution Environment (TEE) deployed on ad tech's AWS account that, upon receiving an aggregation request, retrieved a private key from the coordinators' AWS accounts, decrypted aggregatable report data, processed it, and generated summary reports. Once ready, the summary report was retrieved by the ad tech in order to use it for advertising purposes.

When a summary report was released, the ad tech learned information about users whose contributions were included in the summary report. A goal for the system was to provide a framework that supported differential privacy; as such, the Aggregation Service used a set of tools and mechanisms in order to quantify and limit the amount of information revealed about any individual user, avoiding any form of fingerprinting.

At the time of writing, the Aggregation Service was designed and built around AWS cloud infrastructure. Nevertheless, Google aimed to add support for the more cloud platforms such as Google Cloud in the nearest future.

As stated within the documentation provided by Google, the following principles had to be guaranteed²:

- Uphold The Privacy Sandbox design goals for privacy protections in Attribution Reporting. Specifically, we intend to provide appropriate infrastructure for noise addition and aggregation, aligned with the long-term goal of differential privacy.
- Prevent inappropriate access to raw attributed aggregatable report data or other intermediate data through technical enforcement.
- Allow ad techs to retain control over the data they've collected and access noisy aggregated data without sharing their data with any third party.
- Support flexible, scalable, and extensible aggregation strategies and on-demand access to the aggregation infrastructure, so that ad techs can choose when and how often to generate summary reports.
- Provide open and transparent implementations for any infrastructure outside of the client.

2. Aggregation Service for the Attribution Reporting API Documentation -https://github.com/WICG/attribution-reporting-api/blob/main/AGGREGATION_SERVICE_TEE.md#proposed-design-principles



The software architecture, as described in the documentation, seemed to adhere to the design principles listed above. Few issues were discovered that could potentially undermine Privacy Sandbox Aggregation Service's security posture.



Architecture

Privacy Sandbox Aggregation Service architecture was composed by a significant number of linked systems. The whole service design could be split in 3 different parts, with each part running on a different AWS account.

The following list breaks down the account and trust division:

- One AWS account for ad tech
- One AWS account for Coordinator 1
- One AWS account for Coordinator 2

At the time of writing, the Privacy Sandbox Aggregation Service was designed to run on Amazon Web Services (AWS) cloud environment as this allowed Google to deploy Nitro Enclaves, Trusted Execution Environments (TEEs) that guaranteed the complete opacity of processing within Enclaves to any stakeholder. This concept guaranteed secure processing and user privacy. In the near future, Google plans to expand this service to other major cloud providers.

For the architecture in scope, Google took care of provisioning the Trusted Execution Environment (TEE) software to ad tech and coordinators in the form of Docker images that would be later deployed within an EC2 Instance. Additionally, all stakeholders involved in the process would receive from Google all the guidelines and Terraform code needed to set up the service within their AWS accounts.

After a careful review of the client documentation and clarification from calls, the following architecture diagram was created, which details the components, trust boundaries and communication paths that make up the environment. The client software and ad tech (reporting origin) were independent of the environment and were not in scope.



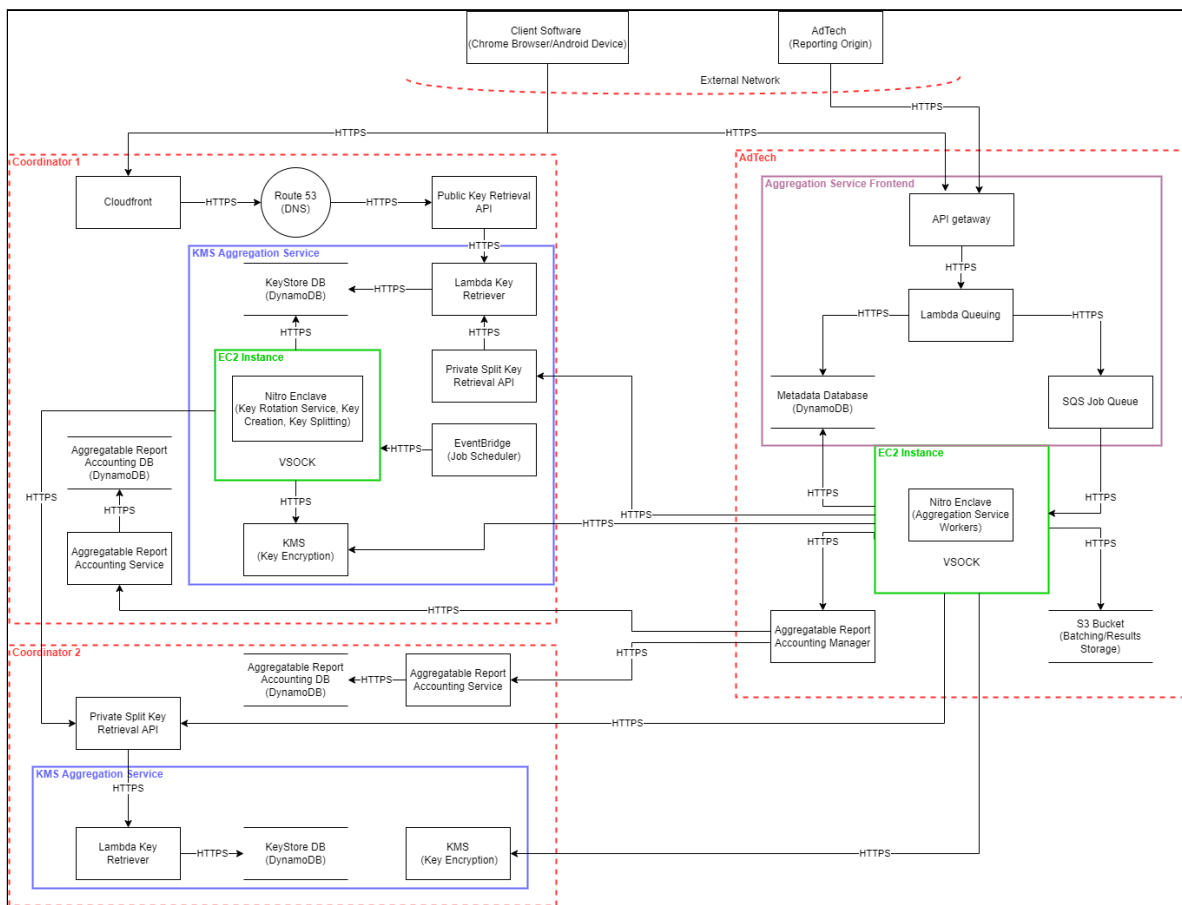


Figure 1: Privacy Sandbox Aggregation Service Data Flow Diagram

A description of each component is listed below:

External Network

- **Client Software:** browser or operating system with the capability to log aggregatable report data and send it encrypted for later processing. This part of the architecture was out of scope. At a high level, the client software reached out to Cloudfront in order to retrieve public keys, encrypted the user aggregatable report data, and sent it to the ad tech, which would later process it using the Aggregation Service.
- **Ad Tech (Reporting Origin):** the ad tech entity that retrieved the report of the aggregatable report data from an S3 Bucket within the ad tech AWS account. This element was not in scope.

Coordinator 1

This was the entity responsible for providing browsers with public keys for aggregatable reports encryption and for generating the private key, needed from the ad tech to decrypt and process aggregatable reports within the Nitro Enclave.

- **Cloudfront:** the browser or operating system communicated with this service in order to retrieve the public key needed to encrypt the aggregation data. Cloudfront checked the cache and, if the key was present, returned it to the browser. If the key was not present, or it had expired, it relayed a request through a Route 53 DNS to the HTTP API responsible for the public key retrieval.
- **Route 53 (DNS):** a domain name system that provided latency-based routing and health checks for a set of redundant APIs. In case of failure, it redirected traffic to alternative Availability Zones or Regions to achieve high availability.



-
- **Public Key Retrieval API:** API that the Client Software (Browser or Operating System) used to get public keys for encryption. The API executed the Lambda Key Retriever function responsible for pulling the public key from the DynamoDB database. The API was copied in a number of different regions and was relying on Route 53 to allow an active-active setup and failover.
 - **Lambda Key Retriever:** this Lambda function was called either by the Public Key Retrieval or Private Split Key Retrieval API. It queried the requested key from the KeyStore (DynamoDB). The key was retrieved from the database and returned to the API which handed it back to the requestor.
 - **KeyStore DB (DynamoDB):** this database stored key-related data that could be requested by Lambda Key Retriever only. The data stored within the database was encrypted.
 - **Private Split Key Retrieval API:** API used from the ad tech's Nitro Enclave in order to get the first part of the split private key. This was necessary to decrypt and process the aggregation data. As for the Public Key Retrieval API, it was relying on Lambda Key Retriever function in order to pull the requested key from the KeyStore.
 - **EventBridge (Job Scheduler):** a job scheduler which triggered a Lambda function within the Trusted Execution Environment that would create and rotate the private key.
 - **KMS Aggregation Service:** service composed of a set of elements, responsible for public and private key creation, manipulation and secure storage.
 - **EC2 Instance:** had the capability to host the Nitro Enclave on dedicated resources that were associated with this instance. The main function of the EC2 Instance was to execute the Nitro Enclave Docker image and proxy the communication from the Enclave to the services that it needed to access, as the Enclave itself was not allowed to communicate with anything other than the EC2 Instance over VSOCK.
 - **Nitro Enclave (Key Rotation Service, Key Creation, Key Splitting):** a Trusted Execution Environment (TEE) that offered an isolated compute environment to protect and securely process highly sensitive data with no computation at runtime observable by any party. This served the purpose of creating a private key, splitting it in half, and securing it by using the KMS service. Additionally, the Enclave was responsible for exchanging and verifying the second half of the key with Coordinator 2. This Trusted Execution Environment was not exposed externally; every communication was performed with the EC2 Instance through the VSOCK protocol. The EC2 Instance was then responsible for communicating with any of the involved services. More details about the Nitro Enclave could be found in the *Nitro Enclave* section below.
 - **Key Management Service (KMS):** this resource helped with the encryption of Private Key split part shared with Coordinator 2. Additionally, it helped for data decryption within ad tech Enclave.
 - **Aggregatable Report Accounting Service:** a distributed service that tracked the number of times certain aggregatable report data had been used to generate aggregated reports, informing the ad tech if this number exceeded a certain threshold. The service used atomic distributed transactions between the Aggregatable Report Accounting Services running on each Coordinator's instances to keep report accounting data state in sync and to detect any form of data tampering. The Aggregatable Report Accounting Service instance was controlled by the Aggregatable Report Accounting Manager hosted in ad tech AWS account. Additionally, the Aggregatable Report Accounting Service was relying on an Aggregatable Report Accounting database that was being used to store report accounting related data.



-
- **Aggregatable Report Accounting DB (DynamoDB):** database that stored report accounting related data. This database was replicated exactly on each of the involved Coordinators' instances. The database consistency was guaranteed by the Aggregatable Report Accounting Service that kept the data always synced between all the replicated storage instances. The data stored within the database was encrypted.

Coordinator 2

This entity's architecture was simpler than Coordinator 1 as it was composed of fewer services. It was responsible for storing the second part of the split private key generated by Coordinator 1.

- **Lambda Key Retriever:** this Lambda was called by the Private Split Key Retrieval API for querying the requested key from KeyStore (DynamoDB). The key was retrieved from the database and returned to the API which handed it back to the requestor.
- **KeyStore DB (DynamoDB):** this database stored key-related data that could be requested by Lambda Key Retriever only. The data stored within the database was encrypted.
- **Private Split Key Retrieval API:** This API was being used to serve two main purposes. The first was to exchange with Coordinator 1's Enclave the KMS symmetric key that was being used to share the second part of the split private key. The second purpose was to hand to ad tech Enclave the second part of the split key needed to decrypt and process the aggregation data. This API relied on a Lambda Key Retriever function in order to pull the requested key from the KeyStore.
- **KMS Aggregation Service:** service composed by a set of elements responsible for the manipulation and secure storage of the second part of the split private key.
- **Key Management Service (KMS):** this resource helped with encryption and decryption of the second part of the split private key data sent by Coordinator 2. Additionally, it was used to generate the symmetric key that was shared between Coordinator 1 and Coordinator 2 in order to encrypt and exchange private key split parts.
- **Aggregatable Report Accounting Service:** a distributed service that tracked the number of times certain aggregatable report data had been used to generate aggregated reports, informing the ad tech if this number exceeded a certain threshold. The service used atomic distributed transactions between the Aggregatable Report Accounting Services running on each Coordinator's instances to keep report accounting data state in sync, and to detect any form of data tampering. The Aggregatable Report Accounting Service instance was controlled by the Aggregatable Report Accounting Manager hosted in ad tech AWS account. Additionally, the Aggregatable Report Accounting Service was relying on an Aggregatable Report Accounting database that was being used to store report accounting related data.
- **Aggregatable Report Accounting DB (DynamoDB):** database that stored report accounting related data. This database was replicated exactly on each of the involved Coordinators' instances. The database consistency was guaranteed by the Aggregatable Report Accounting Service that kept the data always synced between all the replicated storage instances. The data stored within the database was encrypted.

Ad Tech

It hosted the Nitro Enclave that would process aggregatable report data and create an output report.

- **API Gateway:** this API gateway received the batch of aggregatable report data that was batched by ad tech. The data received was forwarded to the Lambda Queuing function in



order to be aggregated afterwards. Additionally, this API was used by ad tech to request aggregation processes and to query for the current status of ongoing jobs.

- **Lambda Queuing:** Lambda function that was responsible for queuing batched aggregatable report data into SQS Queue to be processed later within the Nitro Enclave. Additionally, this function stored request status information related to each aggregation request into a DynamoDB Metadata Database.
- **Metadata Database (DynamoDB):** stored encrypted metadata associated to each aggregation requests such as the current status of processing. This was periodically updated by the Nitro Enclave based on the state of jobs execution. Metadata storage served the function of preventing requests from having duplicate job request keys, which was a requirement of the aggregation service. The data stored within the database was encrypted.
- **SQS Job Queue:** batched aggregation requests of aggregatable report data were queued here in order to be processed afterwards within the Nitro Enclave. The queue policy in place was first in, first out (FIFO).
- **S3 Bucket (Batching/Results Storage):** the designed location for storing plain-text aggregated reports with added noise, in order to grant stronger privacy of data.
- **EC2 Instance:** had the capability to host the Nitro Enclave on dedicated resources that were associated with this instance. The main function of the EC2 Instance was to execute the Nitro Enclave and proxy the communication from the Enclave to the services that it needed to access, as the Enclave itself was not allowed to communicate with anything other than the EC2 Instance over VSOCK.
- **Nitro Enclave:** this Enclave was a Trusted Execution Environment, where confidential and sensitive data could be processed securely. This Enclave processed the aggregatable reports by polling for jobs from the SQS Job Queue. In order to do so, the Enclave retrieved the encrypted split parts of the Private Key from both coordinators and, with the help of coordinators' KMSs, retrieved the information to decryption key's split parts. Once retrieved, private key splits were combined in order to obtain the key needed to decrypt aggregatable report data. Afterwards, the Enclave processed requests aggregating data using differential privacy. This led to the production of a final report that was then stored with added noise within the S3 Bucket. More details about the Nitro Enclave can be found in the *Nitro Enclave* section below.
- **Aggregatable Report Accounting Manager:** the Aggregatable Report Accounting Service's client that was responsible for executing report's accounting consumption operations and propagating them to all Coordinators' Aggregatable Report Accounting Service. It coordinated all the Aggregatable Report Accounting operations through the distributed Aggregatable Report Accounting Service instances.

Nitro Enclave

Privacy Sandbox Aggregation Service's key component was the Nitro Enclave. This was an Amazon Web Services hardened virtual machine that allowed users to create an isolated, trusted and confidential computing environment, which granted high-level security processing and protection for highly sensitive data. This technology used a special Nitro Hypervisor that took care of the execution of the code running within the Enclave, making its run-time totally opaque to any user, administrators included. The Enclave was provided by Google as a Docker image that had to be deployed on a hardened EC2 Instance.



According to the publicly-available AWS documentation³, Nitro Enclaves were providing the following properties which NCC Group assumed to be true for the purposes of this assessment:

- An independent kernel, CPU and memory environment separated from the EC2 Instance hosting the Trusted Execution Environment.
- No direct interactive access from outside the Enclave, even for profiles with full IAM permissions. Additionally, all communications were encrypted and performed with the EC2 Instance through VSOCK.
- KMS integration to enable the decryption of secrets within the Enclave.
- Cryptographic attestation of Nitro Enclaves to verify the identity of the instances.

At the time of writing, Google was responsible for building and providing Nitro Enclave images that would be deployed within stakeholders' EC2 Instances as containerized. Nevertheless, the ad techs were given the option to autonomously build the Aggregation Service from the open source code that was publicly available on Google's GitHub repository⁴.

Upon building them, a cryptographic hash was created, the Platform Configuration Registers (PCR); this hash could be used by any of the stakeholders in order to verify the integrity of the Enclave images against a published codebase. Additionally, any of the parties could audit the full codebase of the Nitro images in use.

IAM's policies for KMS were meant to allow only authorized Enclaves with specific PCRs to execute actions such as decryption of secrets, as the attestation document sent along with the request would contain only authorized PCRs values.⁵

None of the Nitro Enclave image code could be modified and tampered with without prejudicing the functioning of the Enclaves as, upon any image alteration, the PCRs would change, invalidating any eventual IAM policy. This prevented any altered Enclave Image from accessing private key related information required to decrypt the aggregatable report data, that would be later aggregated within the trusted execution environment. The aforementioned PCR checks, along with the unobservability property of the Nitro Enclave, ensured the confidentiality and integrity of the data to be processed.

As a result of the above features and configuration, the chances for a malicious ad tech or Coordinator to be able to compromise the Nitro Enclave and spoof aggregation requests processing were negligible. Nevertheless, for the test environment in scope, NCC Group identified two issues affecting the Nitro Enclave instance:

- As reported in [finding "Lack of VM Image Hardening"](#), the EC2 Instance hosting the Nitro Enclave appeared to lack the hardening requisites that were documented for the service. The EC2 Linux image was observed to contain unnecessary utilities that could potentially allow a malicious user to compromise the EC2 Instance and, eventually, the Nitro Enclave. Additionally, the secure boot option for the Linux image was not set. This made impossible to guarantee that the state of the machine at boot time was indeed safe.
- Another risk was represented by the Nitro Enclave Docker image itself. As reported in [finding "Docker Image with Scan Findings"](#), by using commonly-available scanning tools, it was possible to observe that the container image provided by Google was vulnerable

3. AWS Nitro Enclaves Features - <https://aws.amazon.com/ec2/nitro/nitro-enclaves/features/>

4. Building Aggregation Service Artifacts - <https://github.com/privacysandbox/aggregation-service/blob/main/build-scripts/aws/README.md>

5. Cryptographic Attestation- <https://docs.aws.amazon.com/enclaves/latest/user/set-up-attestation.html>



to a number of Common Vulnerabilities and Exposures (CVEs) which risk rating was placed between medium and high risk. Providing the ad techs with a vulnerable Docker image could cause Nitro Enclaves to be accessed by malicious agents.

Recommendations

- Any EC2 Instance that is meant to host the Nitro Enclave should be hardened according to the documentation. Superfluous utilities should be removed and the secure boot option should be enforced.
- All Docker images provided by Google should use distroless images and be checked properly to uncover common vulnerabilities.

Retest Results

Google's implementation of the recommendations is discussed in the retest sections of [NCC-E004186-JPU](#), [NCC-E004186-R6C](#), and [NCC-E004186-JPU](#).

Aggregatable Report Accounting Service

One of the components involved in the processing of aggregation requests was the Aggregatable Report Accounting Service. This was a service designed by Google whose purpose was to track the number of aggregation requests performed from ad techs. It allowed aggregatable report data retrieval while limiting the amount of user-related information that could be retrieved by ad techs, avoiding any form of fingerprinting and granting user privacy.

Ad techs were assigned an equal number of tokens (Report Accounting) that represented the value of the information that could be collected from the user. For each aggregation request, a number of tokens were consumed according to the estimated value of the data collected. Once all the Aggregatable Report Accounting tokens were consumed, ad techs could not collect and process additional aggregatable report data.

The Aggregatable Report Accounting Service was implemented as a distributed service that used atomic transactions which were shared and synchronized among Coordinators. Additionally, Aggregatable Report Accounting Service instances, through the Aggregatable Report Accounting Manager, were responsible for constantly updating and maintaining the consistency of Aggregatable Report Accounting data within each instance's Report Accounting Database. This prevented any form of tampering as any malicious attempt to modify accounting data would cause inconsistency between the distributed transactions and trigger a roll back of any token consumption.

Aggregatable Report Accounting Service data could not be accessed by the ad tech and the Coordinators, only the Aggregatable Report Accounting Service itself could access and manipulate report accounting data.

The Aggregatable Report Accounting key (ARAK) was one of the critical components of the Aggregatable Report Accounting Service. The ARAK was a hashed string created within the Aggregation worker hosted in ad tech Enclave. This was created combining a set of values generated from the client device such as API endpoint, version, reporting origin (ad tech identity), destination (advertiser domain) and source registration time. Once the hash was created, the ad tech enclave sent it to the Aggregatable Report Accounting Manager. This checked the hash value as a security countermeasure preventing malicious actors from spoofing the aggregation request to impersonate and re-query the data in the name of a different ad tech.



Environment

The totality of the documentation provided by Google was written for AWS resources that, as aforementioned, allowed Google to deploy Nitro Enclaves that granted protection and privacy for users' data.

As stated within the design documents provided, AWS was implicitly trusted given the strong incentives it had to guarantee its terms of service. Scenarios in which the cloud provider could abuse its privileged position in order to compromise or exfiltrate data were considered, but their mitigations were out of scope as the probability was negligible.

Communication

Through the documentation reviewed and interviews held with Google, it was observed that all the communication between the Privacy Sandbox Aggregation Service was performed using HTTPS. The communication between the EC2 Instance and Nitro Enclaves used VSOCK.

Due to the security measure and communication protocols in place, no practical vectors were indentified that would allow a malicious actor to be able to perform a man-in-the-middle attack.

Service Segmentation

When designing infrastructure, careful consideration should be put into how services can be separated in order to achieve the following goals:

- Preventing single points of failure from causing system outages.
- Ensuring that sensitive data is stored in the most secure, heavily audited systems.
- Placing boundaries between unrelated systems to prevent attackers from easily pivoting across networks.
- Limiting the effect of compromise to only the data stored in a single system.

At the time of writing, it was observed that Privacy Sandbox Aggregation Service infrastructure was split between three different AWS accounts: ad tech, Coordinator 1 and Coordinator 2.

The ad tech account hosted the main infrastructure that was responsible for the aggregation process. This was being executed within the Nitro Enclave, a Trusted Execution Environment that made totally opaque any type of data computation to any agent, even to administrators of the ad tech AWS account.

The Coordinator 1 account was responsible for the creation, rotation and key splitting of the Private key needed to decrypt the aggregation requests being processed in the ad tech Trusted Execution Environment. This party was relying on a Nitro Enclave for all the operations involving the Private Key, thus ensuring that nobody could actually observe or retrieve its value in clear-text.

The Coordinator 2 account hosted resources responsible for the storage and exchange of the second part of the encrypted Private Key. This key split may be retrieved by the ad tech account and combined with the first part of the Private key, held by Coordinator 1, in order to decrypt aggregatable report data and process it.

The above segmentation was tactically chosen by Google in order to grant a state-of-the-art level of confidentiality for the user data collected in aggregatable reports. None of the involved parties, even administrators of the respective AWS accounts, were able to retrieve clear-text aggregatable report data.



Logging and Auditing

CloudTrail was enabled for the Public Key Retrieval and Private Split Key Retrieval API while CloudWatch was integrated with the remaining services such as Private Key generation or Aggregatable Report Accounting. This allowed Google to constantly monitor services health through the Privacy Sandbox Aggregation Service.

One of the most critical logs was the Aggregatable Report Accounting Service implementation log. As the report accounting information was directly linked with the amount of data collectable from an ad tech, any event that would change the status of report accounting data would be logged before the action was executed. This log collection allowed the complete recovery of the report accounting data in case of corruption. However, it was observed that no alerting policies were in place for the Aggregatable Report Accounting DB and, in general, for DynamoDB databases. As these were storing critical data, such as public and private key data and Aggregatable Report Accounting entries, the lack of any alert configuration would cause Google to be unaware of situations that could impact on databases' performances and availability. This issue is documented in [finding "Dynamo DB Alerting Not Enabled"](#).

Recommendations

- CloudWatch alarms should be configured for all integrated DynamoDB instances, as described in the DynamoDB documentation.⁶

Retest Results

- The recommendations has been addressed in the retest section of [NCC-E004186-HH4](#).

Confidentiality

The biggest claim for the Privacy Sandbox Aggregation Service was that end user data, collected in aggregatable reports, was protected with strong encryption mechanisms and could not be read by any of the involved ad techs or Coordinators. This granted that identity of individual users was protected. To achieve this objective, a consistent number of security measures were adopted by Google in the service's design.

The content of the aggregation request sent from a generic browser was encrypted with a public key provided by a Coordinator before being sent to the ad tech for processing. This approach prevented ad techs from reading any of the content of the aggregatable report data, granting end-user privacy and data confidentiality. Additionally, the aggregation process of the data was performed from the ad tech within the Nitro Enclave.

This component was responsible for fetching the private key split parts needed to decrypt the content of the aggregation request's payload. The key was split in two encrypted parts that were fetched from two different Coordinators. This key division prevented Coordinators and ad tech from accessing the full private key. Additionally, none of the computation performed by the Nitro Enclave at runtime was observable by any party, administrators included, making the aggregation process completely opaque to everyone.

Only the final report of the aggregation process was stored in clear-text on an S3 Bucket, accessible directly from the correspondent ad tech. Before storing the report, the Nitro Enclave added noise to grant an additional privacy protection layer, as part of the differential privacy framework.

In order to compromise the confidentiality of the Privacy Sandbox Aggregation Service, both Coordinators and the ad tech would have to collude to exchange private key data and

6. Creating CloudWatch alarms to monitor DynamoDB - <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/creating-alarms.html>



decrypt the aggregatable reports being sent to be processed within the Nitro Enclave. Nevertheless, this scenario has been deemed highly unlikely.

Integrity

From the documentation, the Privacy Sandbox Aggregation Service was observed to implement a number of controls that preserved and validated the integrity of the data the service was holding.

As the aggregatable report data sent from the browser was natively encrypted, none of the parties could be able to modify it. The only instance that could decrypt and manipulate the aggregation data would be the Nitro Enclave.

In order to prevent unsupported aggregation jobs from reaching the SQS Queue, a number of fields from the aggregation request payload, such as the reporting origin, were checked. Additionally, the Aggregatable Report Accounting Service was responsible for checking the hash value of the Aggregatable Report Accounting Key, generated within the Enclave, in order to validate the legitimacy of the request.

It should be noted that all of the DynamoDB databases within Privacy Sandbox Aggregation Service were adopting a point-in-time recovery strategy in order to help ensure the consistency of data and avoid any type of discrepancy or, more in general, data loss.

Availability

Privacy Sandbox Aggregation Service was designed around a large number of different components. AWS provided scalability and availability assurances for some managed components such as Lambda, DynamoDB, and S3. However, while the underlying services could have availability assurances, the data within them could not: operator errors or malicious tampering may destroy that data or disrupt access to it and the service had no provisions to fail over to other instances. Additionally, no provision for availability was made for other components such as the EC2 Instances and Nitro Enclaves; any failure in these components would render the application unusable.

As Coordinators were responsible for providing Private Keys, a failure in the Coordinators would result in the impossibility for the ad tech to retrieve the private key, hence avoiding the decryption and processing of aggregation requests. Nevertheless, a number of failure modes would still allow the ad tech to continue accepting aggregation requests that would be kept in the processing SQS queue until the recovery of the failing Coordinator normal status.

As few provisions to ensure availability were made, a failure in Coordinator was considered to be a possible scenario within the current architecture.

Access Control

Google's design documentation for the Privacy Sandbox Aggregation Service showed concern for proper authentication and access control policies.

All application components were hosted within AWS. This allowed them to use various AWS mechanisms to manage access, including IAM Policies and VPC Security Groups. The application design showed that fine-grained controls were in place and that best practices for credential management were being followed.

Data aggregation was processed in Nitro Enclaves within the ad tech AWS account. Due to the nature of these Enclaves, no party could access or spoof the computation of aggregatable report data within the trusted execution environment, even account administrators. ad techs were only able to access final summary reports which contained aggregated data with added noise, according to the differential privacy framework.



The application also employed cryptographic protection to ensure that only authorized parties could make requests to the system or access data. The private key used to decrypt and process aggregation data within the ad tech enclave was generated within another Nitro Enclave in the Coordinator 1 AWS account before being split in two parts and stored in KMS in two separate Coordinator accounts. The secrecy of the key split values exchanged by the Coordinators was there assured by the same Trusted Execution Environment and attestation mechanism used for aggregation calculation in the ad tech enclave, ensuring that no party could access the private key or decrypt data.

Any Coordinator member willing to access the AWS console and modify any service participating in the Privacy Sandbox Aggregation Service required the approval of another team member. This prevented any authorized internal users to actually conduct malicious modification to Privacy Sandbox Aggregation Service's components hosted in Coordinators' AWS accounts.

As reported in the *Nitro Enclave* section above, KMS access to the trusted execution environment was implemented with IAM policies that, by using cryptographic attestation, granted that only a set of authorized Enclaves could be authorized to access key values that would be used for the decryption and processing of highly sensible data within the Enclave.

It should be noted that Google provided the ad techs with the proper Terraform and cloudformation templates in order to deploy securely the aggregation service within their AWS account. Nitro Enclave images were already provided as secured to stakeholders. Nevertheless, additional access control configurations for services outside the trusted execution environment such as S3 Bucket, SQS Queue and APIs' access permissions, were up to ad techs' discretion.



6 Holistic Attacker-Modeled Pentest

Intro

This section documents the holistic attacker-modeled penetration testing of the final design and implementation of the Privacy Sandbox Aggregation Service. It is performed from the perspective of a malicious ad tech client attempting to willfully subvert its security features and design goals. NCC Group tested based on the architecture models from phases one and two of the assessment. After the architecture and the AWS services configurations reviews, NCC Group attempted dynamic testing to identify privilege escalation attacks, data access, data leakage and access controls misconfigurations.

NCC Group assumed the perspective of a malicious ad tech client attempting to manipulate the system and the AWS cloud environment. The consultants tried to bypass the security policies and gaining access to protected data either directly or indirectly. These activities included verifying security boundaries, authentication, access controls, and settings in Amazon EC2, S3, DynamoDB, SQS, Nitro enclave and other related services. The focus was on identifying potential ways to undermine the service in several high level areas described in the following sections.

Authentication, Authorization and Network Security

Identity and Access Management (IAM) settings and policies were reviewed to look for privilege escalation paths and overly permissive policies. The virtual machine hosting the Nitro enclave had no external IP address and was well protected by its network security groups, so it was not reachable from external and internal network locations. Both the Amazon machine image (AMI) used by the virtual machine and the Docker image used by the Nitro enclave were scanned and assessed for vulnerabilities and unhardened configurations. Two improvements were suggested in [finding "Lack of VM Image Hardening"](#) and in [finding "Docker Image with Scan Findings"](#). However, NCC Group notes that choosing and running the AMI is ad tech's responsibility.

Both in-transit and at-rest encryption were found to be applied on all services. Accessing and querying the DynamoDB database and the S3 bucket were impossible without authentication. The IAM policies enforced access controls and blocked all public access. The roles attached to the Lambda functions and the VM showed no possible privilege escalation, but one role is able to list more parameters than required ([finding "IAM Role Assigned with Excessive Permission"](#)). NCC Group observed that the Lambda function did not have code signing enabled ([finding "Lambda Function Without Code Signing"](#)), which is also ad tech's responsibility.

Credentials and Key Protection

The consultants interacted with the API endpoints as authenticated and unauthenticated users to retrieve encryption keys, access keys and tokens. The endpoints were well protected and could not be accessed without appropriate permission. In addition, the KMS keys had no default policies and API gateway access required authentication.

While NCC Group gained access to an access token for an IAM role via Instance Metadata Service (IMDS) v1 by simulating a server-side request forgery attack, it was not possible to gain access to any encryption or decryption keys. The role had permissions only to a limited set of resources and is discussed in more details in [finding "EC2 Instance Metadata Service Version 1 In Use"](#).

Denial of Service Attacks

Stopping the enclave without root access was attempted but found to be impossible. If the enclave stopped, the SQS service would queue all existing jobs and trigger a CloudWatch alarm with an SNS action then the enclave continue processing them after a restart. The API



endpoints did not process any other data structures or malformed requests and always returned “Not Found” messages.

The API gateway and Lambda functions had active rate limiting enabled and configured. Submitting a large number of requests in a short time did not affect application performance because the SQS service queued the incoming requests and rate limiting prevented overloading the services.

Side Channel Attacks

Requests towards API endpoints available for ad tech were measured for time differences, but results were inconclusive.

Enclave and Environment Protection

Attempts were made to access enclave via the console and searching the output for any sensitive information, but no sensitive information was leaked or logged. NCC Group was additionally unable to find sensitive information in process memory dumps. The `vsock` configuration file was reviewed and `vsock` traffic was analyzed for any unencrypted communication, but only encrypted communication was configured.

NCC Group created and ran enclave images that used the running enclave image as a base image, but could not gain access to any keys or data, as a result of the KMS key policy condition (`kms:RecipientAttestation:PCR0`) and the absence of hard-coded credentials. The review of the enclave image build process and the vulnerability scans showed some results as discussed in [finding "Docker Image with Scan Findings"](#).

Sensitive Data Logging Features Review

Reviewing and searching the enclave logs (`/var/log/nitro_enclaves/`), enclave console logs (`nitro-cli console`) and CloudTrail logs including nitro attestation (key access activities) revealed no sensitive information or any trace of sensitive information logging.



7 Web Service Review

This section documents the web services that NCC Group has reviewed for this assessment. NCC Group followed a general testing methodology for all features, which included a static code review and assessing relevant code differences for changes to features from phase one of the assessment. Dynamic testing was also employed to identify issues that might lead to privilege escalation, data leakage, or bypass of designed access controls.

Overview

The Cloud Secure Control Plane provides the frontend service to create jobs, which are then processed asynchronously by the Data Processing Service, also known as the Aggregation Service. The frontend service offers the `createJob` API which allow operators to create a job for the Aggregation Service. When the `createJob` API is called, it retrieves aggregatable reports from a file storage in AWS S3 then stores the job details in a database, and adds a message to a queue. The Aggregation Service then listens for these messages and proceeds to process the job to generate a summary report. The `getJob` API allows the operator to retrieve the status of the job. Below is the workflow diagram provided from SCP Frontend Service documentation.

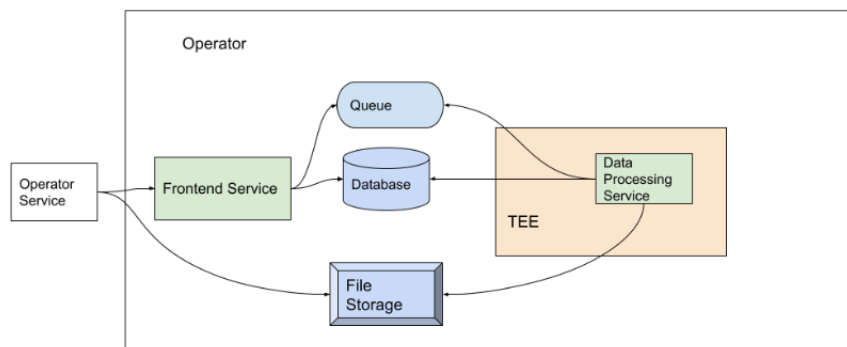


Figure 2: Web Service Workflow

Web Service Testing

The consultants performed dynamic testing of all APIs in scope to ensure that the API correctly handles any malformed inputs including unexpected string encodings, integer boundary conditions, excessively large inputs, empty fields, missing fields, and unexpectedly typed fields. Each API was tested to verify that they correctly implement the expected authorization behavior and reject all unauthenticated requests. This was accomplished by a series of positive and negative validation test cases for authorization. Furthermore, during the assessment, the consultants tested for vulnerabilities that commonly affect APIs such as injection flaws, privilege escalation flaws across instances of the Aggregation Service and other logic flaws. From the series of test cases, the consultants concluded that the in-scope APIs have a strong security posture and that the appropriate protections are in place to defend against common threats against APIs.



CreateJob API

The API endpoint for creating a job, located at `https://<api-gateway>/stage/v1alpha/createJob`, accepts JSON data in the request body and starts a job based on the provided parameters. The API is not publicly available and uses AWS API Gateway to manage authorization, validation, and routing of requests. During the assessment, the consultants examined the potential risks of the API and found that any attempts at exploitation would likely only result in self-inflicted damage, as the API can only be accessed by authorized users or services. This is based on the assumption that these authorized operators have not been compromised. However, there were a few unlikely attack vectors that an attacker could potentially abuse. An authenticated attacker can manipulate the “write location” parameter (`output_data_blob_prefix`) to overwrite existing data. The application does not change the specified file name, allowing the attacker to potentially overwrite previous output files or even encrypted source data. More information about this finding could be found in [finding "Lack of Overwrite Controls in S3"](#).

Ingested files for CreateJob API

The aggregatable reports is formatted as an `.arvo` file which was provided within an S3 bucket during testing. The consultant examined the series of `batch.arvo` files to ensure that there were no sensitive information being leaked. Upon inspection of the `batch.arvo` file using `AvroViewer`, the data is presented in JSON format. The “payload” is a key-value pair which contains a base64 encoded Concise Binary Object Representation (CBOR) payload and the other key value pair such as `key_id` and `shared_info` contains no sensitive information. The summary reports generated by the `CreateJob` API outputted similar results. The consultants attempted to create a malicious `.arvo` file and invoked the `CreateJob` API to determine if they could elicit an error message or gain insight into potential ways to exploit the Aggregation Service. However, the service effectively identified and rejected the unfamiliar `.arvo` files, indicating that robust security measures are in place.



8 Source Code Review

Overview

The Privacy Sandbox Aggregation Service consists of two components: the Secure Control Plane (SCP) and the Data Plane (DP). The SCP provides a secure execution environment and manages the DP at scale, hosting the Aggregatable Report Accounting Service (ARAS) as well. The SCP execution system enables the service to be utilized on any supported cloud provider without modification to the DP, and it communicates with the DP through the Control Plane Input/Output (CPIO). The DP, implemented as a Docker container, handles business logic for specific computations, such as managing cryptographic keys, buffering requests, tracking the aggregatable report accounting, accessing storage, and more. It also hosts the CPIO. The CPIO acts as a pass through channel when communicating with coordinators. While data transmitted between clients and the DP passes through the SCP, the data remains encrypted and the SCP does not have access to the decryption keys.

NCC Group performed a source code assisted security assessment, focused on the SCP and DP. This review was performed by two (2) consultants between the days of December 5th and January 16th, 2023, focused on the following aspects of the applications:

- Authentication and Authorization
- Input Validation Mechanisms
- Privacy Implementation
- Split Key Retrieval Implementation
- Decryption Implementation
- Aggregatable Report Accounting Service

Front-end API Gateway

Front-end Authorization

The front end services rely on AWS API Gateway (APIGW) to perform authorization checks. Before dispatching API calls to their respective handlers, the front-end AWS API Gateway (APIGW) verifies request authentication and authorization using the `ApiGatewayHandler` class". The `ApiGatewayHandler` class doesn't provide specific implementation for authorization, but it provides a method named `authorizeRequest` that can be overridden by subclasses.

```
/**
 * Authorizes the request. Simply returns void if the request authorization passes. Throws a
 * ↳ ServiceException if an error occurs. Unless overridden this does nothing, meaning that
 * ↳ unless a subclass implements this then authorization is disabled. This should be
 * ↳ overridden in subclasses that need authorization.
 */
protected void authorizeRequest(
    Request request, APiGatewayProxyRequestEvent apiGatewayProxyRequestEvent)
    throws ServiceException {}
```

In this example, the authorization mechanism for `createJob` is actually being handled within the `CreateJobApiGatewayHandler`. The `CreateJobApiGatewayHandler` class acts as a handler for incoming API Gateway requests in AWS Lambda. The code snippet below demonstrates, when the function is triggered by an API Gateway request, the `handleRequest` method is called, which first calls `toRequest` to convert the API Gateway request into a specific request object that the service recognizes. The method then calls `authorizeRequest` to



authorize the request, and finally `processRequest` to handle the request and return a response.

```
/** Handles incoming request. */
@Override
public APIGatewayProxyResponseEvent handleRequest(
    APIGatewayProxyRequestEvent apiGatewayProxyRequestEvent, Context context) {
    try {
        Request request = toRequest(apiGatewayProxyRequestEvent, context);
        authorizeRequest(request, apiGatewayProxyRequestEvent);
    }
}
```

CreateJobApiGatewayHandler

The `CreateJobApiGatewayHandler` class is used to handle requests to create a new job. It does this by converting the incoming API Gateway request into a `CreateJobRequest` object, which it then passes to the front-end services to create the job. The response from the front-end services is then converted into an API Gateway response and returned to the client. The class also has a method to validate the properties of the `CreateJobRequest` object to ensure that all required fields are present before processing the request.

GetJobApiGatewayHandler

The `GetJobApiGatewayHandler` class is used to handle GET requests to the API Gateway. It receives a `job_request_id` parameter and retrieves information about the job specified by that user input identifier. The class takes the job request ID from the query parameters of the request and creates a `GetJobRequest` object with it. It then uses the front-end service object's `getJob` method to retrieve the job with the specified ID.

Web Services Input Validation Mechanisms

After conducting a thorough review and analysis of the input validation mechanisms in place for the web services, NCC Group did not identify any significant issues. The Lambda functions appear to be properly implemented and secure. The consultants did not identify any vulnerabilities or weaknesses that would compromise the integrity of the input validation process.

In particular, NCC Group reviewed both the `GetJobApiGatewayHandler` and `CreateJobAPIGatewayHandler` classes, which handle job retrieval and creation respectively. These classes implement methods to convert incoming requests, process them, and convert the response. Additionally, the classes validate the properties of the `GetJobRequest` and `CreateJobRequest` objects, respectively, and throw a `ServiceException` in case any properties are missing or invalid.

Furthermore, the consultants validated the inputs for the `createJob` and `getJob` APIs against the requirements listed below in the Aggregation Service API Documentation and did not identify any discrepancies or issues.

```
{
  // Unique identifier. Length must be 128 characters or less.
  // Legal characters are ASCII letters (a-z, A-Z), ASCII
  // numbers (0-9), and the following ASCII punctuation
  // characters !"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~
  "job_request_id": <string>,
  // input file bucket and path in bucket, can be prefix for
  // sharded inputs
  "input_data_blob_prefix": <string>,
  "input_data_bucket_name": <string>,
  // output file bucket and path in bucket, can be prefix for sharded outputs
  "output_data_blob_prefix": <string>,
```



```

// output data bucket
"output_data_bucket_name": <string>,
"job_parameters": {
  // location of pre-listed aggregation buckets
  "output_domain_blob_prefix": <string>,
  "output_domain_bucket_name": <string>,
  // reporting URL
  "attribution_report_to": <string>,
  // [Optional] differential privacy epsilon value to be used for this job. 0.0 <
  ↳ debug_privacy_epsilon <= 64.0. The value can be varied so that tests with different
  ↳ epsilon values can be performed during the origin trial.
  "debug_privacy_epsilon": <floating point, double>
}
}

```

Aggregation Service

Workflow of Aggregation Service

The `ConcurrentAggregationProcessor` is a job processor that uses in-memory aggregation. It starts by reading the job request invoked by the `createJob` API. The service finds the data shards of the reports and reads the data from each data shard using the `BlobStorageClient` and `AvroReportsReaderFactory` classes in an asynchronous manner. The processor then decrypts and validates each report asynchronously using the `ReportDecrypterAndValidator` class to ensure the security and integrity of the data.

After that, the processor asynchronously performs the aggregation and noising of the data using the `AggregationEngine` and `NoisedAggregationRunner`, before sending a request to Aggregatable Report Accounting Service (ARAS) to consume report accounting budget. If the ARAS returns that the reports were previously fully processed, an exception is thrown. Any invalid reports that failed to be decrypted and validated are collected for further investigation. Finally, the aggregation service logs the result and returns it to the client. If the user requested a debug run, the service also adds debug information to the final result for analysis. Aggregation service also limits the results to the input domain provided by the user.

Data Integrity and Confidentiality

To ensure the security of the integrity and confidentiality of the data, the service uses the `ReportDecrypterAndValidator` class to decrypt and validate each report, which ensures that the data is both secure and has not been tampered with. In the Java code snippet below, the `decryptShard` method is called with a job context, list of encrypted reports, and shard index. The method creates a `Stopwatch` for measuring the elapsed time until the shard is decrypted. It then maps each report to the result of decrypting and validating the report using the `reportDecrypterAndValidator.decryptAndValidate(report, ctx)` method, which decrypts the report and checks that the report is valid. Finally, the method collects the results of the decryption and validation into an `ImmutableList` and returns that list.

```

private ImmutableList<DecryptionValidationResult> decryptShard(
    Job ctx, ImmutableList<EncryptedReport> shard, long shardIndex) {
    Stopwatch decryptionStopwatch =
        stopwatches.createStopwatch(String.format("shard-decrypt-%d", shardIndex));
    decryptionStopwatch.start();
    ImmutableList<DecryptionValidationResult> results =
        shard.stream()
            .map(report -> reportDecrypterAndValidator.decryptAndValidate(report, ctx))

```



```

        .collect(toImmutableList());
    decryptionStopwatch.stop();
    return results;
}

```

Decryption Process

The `decryptAndValidate` method first decrypts the encrypted report using the `RecordDecrypter`'s `decryptSingleReport` method, and then performs various validations on the decrypted report using a set of `ReportValidator` objects. If the decryption and validation processes are successful, the decrypted report is returned. Otherwise, a list of errors is returned, ensuring that only secure and valid reports are processed by the service.

The decryption process happens in the `decryptSingleReport` method of the `DeserializingReportDecrypter` class demonstrated in the code block below. This method takes an `EncryptedReport` object as input, then goes through the following steps:

1. Deserialize the `sharedInfo` from the `EncryptedReport` object using the `sharedInfoSerdes` object.
2. The `DecryptionCipherFactory` inspects the provided `{@code EncryptedReport}` for the key used to decrypt, retrieves the key it needs from the `{@code DecryptionKeyService}`, and constructs a decryption cipher using that key. Essentially, the service uses the `HybridDecryptionCipherFactory` object to create a `DecryptionCipher` object for the `EncryptedReport` object.
3. Use the `DecryptionCipher` object to decrypt the payload of the `EncryptedReport` object into plaintext bytes.
4. Deserialize the plaintext bytes into a `Payload` object using the `payloadSerdes` object.
5. Return a new `Report` object that contains the deserialized `Payload` object and `SharedInfo` object.

```

@Override
public Report decryptSingleReport(EncryptedReport encryptedReport) throws
↳ DecryptionException {
    try {
        // Deserialize the sharedInfo
        Optional<SharedInfo> sharedInfo = sharedInfoSerdes.convert(encryptedReport.sharedInfo());
        if (sharedInfo.isEmpty()) {
            throw new DecryptionException(
                new IllegalArgumentException(
                    "Couldn't deserialize shared_info. shared_info was: "
                    + encryptedReport.sharedInfo());
        }
        // Decrypt the payload to plaintext bytes
        DecryptionCipher decryptionCipher =
            decryptionCipherFactory.decryptionCipherFor(encryptedReport);
        ByteSource decryptedPayload =
            decryptionCipher.decrypt(
                encryptedReport.payload(), encryptedReport.sharedInfo(),
                ↳ sharedInfo.get().version());
        // Deserialize the payload
        Optional<Payload> plaintextPayload = payloadSerdes.convert(decryptedPayload);
        if (plaintextPayload.isEmpty()) {
            throw new PayloadDecryptionException(
                new IllegalArgumentException("Decrypted payload could not be deserialized"));
        }
        return Report.builder()

```



```

        .setPayload(plaintextPayload.get())
        .setSharedInfo(sharedInfo.get())
        .build();
    } catch (PayloadDecryptionException | CipherCreationException e) {
        throw new DecryptionException(e);
    }
}
}
}

```

In the event that the wrong encryption or key is used, or if the report has been tampered with, the `decryptAndValidate` method of the `ReportDecrypterAndValidator` class will fail to properly decrypt the report. This failure will trigger a `DECRYPTION_ERROR`, which gets reflected in the body response of the `getJob` API specifically the status parameter. This error indicates that there is a problem with the encryption key or the integrity of the report, and the data cannot be trusted.

Coordinator Key Retrieval

The Coordinator Key Retrieval used for decryption is accomplished within the `MultiPartyDecryptionKeyServiceImpl` class, which is responsible for managing the decryption keys. The `MultiPartyDecryptionKeyServiceImpl` class has two `EncryptionKeyFetchingService` members, `coordinatorAEncryptionKeyFetchingService` and `coordinatorBEncryptionKeyFetchingService`, that are responsible for fetching the split key located in the coordinator's AWS Key Management System(KMS). The `MultiPartyDecryptionKeyServiceImpl` class also maintains a cache of `HybridDecrypt` objects that it creates, which it uses to return the decrypter for a given key when `getDecrypter` is called.

When the `getDecrypter` method is called, it first checks the cache for the decrypter. If it is not found in the cache, it calls the `createDecrypter` method which fetches the primary encryption key from the KMS using `coordinatorAEncryptionKeyFetchingService`. There are two types of keys that can be fetched, single-party or multi-party.

- If the key is a single-party key, it uses the `coordinatorAAeadService` instance to create the decrypter and returns the key.
- If the key is a multi-party key, it fetches the secondary encryption key from the KMS using `coordinatorBEncryptionKeyFetchingService`, and uses the `coordinatorBAeadService` to create the decrypter and returns the key.

The following code shows the practical implementation of creating a `HybridDecrypt` decrypter object from two encryption keys, `encryptionKeyA` and `encryptionKeyB`, which are obtained from Coordinator A and Coordinator B respectively. First, the service retrieves the encrypted key material from the `EncryptionKey` objects by calling the `getOwnerKeyData()` method. Then, it uses an AEAD (Authenticated Encryption with Associated Data) object, obtained from the `coordinatorAAeadService` or `coordinatorBAeadService` instance, to decrypt the key material. The decrypted key material is called "splitA" and "splitB" respectively.

```

private HybridDecrypt createDecrypterSplitKey(
    EncryptionKey encryptionKeyA, EncryptionKey encryptionKeyB)
    throws GeneralSecurityException, IOException {
    // Split A.
    var encryptionKeyAData = getOwnerKeyData(encryptionKeyA);
    var aeadA = coordinatorAAeadService.getAead(encryptionKeyAData.getKeyEncryptionKeyUri());
    var splitA =
        ↳ aeadA.decrypt(Base64.getDecoder().decode(encryptionKeyAData.getKeyMaterial()), new
        ↳ byte[0]);
}

```



```
// Split B.
var encryptionKeyBData = getOwnerKeyData(encryptionKeyB);
var aeadB = coordinatorBAeadService.getAead(encryptionKeyBData.getKeyEncryptionKeyUri());
var splitB =
    aeadB.decrypt(Base64.getDecoder().decode(encryptionKeyBData.getKeyMaterial()), new
        ↳ byte[0]);\
```

Afterwards, the service uses the `KeySplitUtil` class to reconstruct the keyset handle by calling the `reconstructXorKeysetHandle` method, passing the splits A and B. This combines the two splits into one keyset handle. Finally, it returns the `HybridDecrypt` object, a decrypter class, by calling the `getPrimitive` method on the keyset handle.

```
// Reconstruct.
var keySetHandle =
    KeySplitUtil.reconstructXorKeysetHandle(
        ImmutableList.of(ByteString.copyFrom(splitA), ByteString.copyFrom(splitB)));
return keySetHandle.getPrimitive(HybridDecrypt.class);
}
```

Report Validation

The consultants identified a number of validation checks performed by the service to ensure the integrity and security of the reports. These checks include:

- Verifying that the `attribution_report_to` job parameter in a job is not empty or blank.
- Ensuring that the report's debug mode is checked.
- Confirming that the report's reporting origin is a valid domain.
- Comparing the report's reporting origin to the attribution report to provided in the Aggregation Request.
- Ensuring that the report's operation is an accepted value.

Additionally, the service detects and handles errors that may occur while reading the report using `ConcurrentShardReadExceptions` and `DomainReadExceptions` classes. When these errors occur, the service throws an `AggregationJobProcessException` and sets the error message to `INPUT_DATA_READ_FAILED`, which is returned to the status parameter of the `getJob` API. This indicates that there were problems with reading the reports.

Invalid Test Management

Large numbers of invalid reports could potentially lead to inaccurate or biased results in the final aggregation. Collecting too many invalid reports can also lead to the use of excessive memory and processing resources, potentially slowing down the overall aggregation process. To handle this, the service collects invalid reports and limits the number of invalid reports collected to 1000 reports maximum.

```
javaprivate static final int MAX_INVALID_REPORTS_COLLECTED = 1000;
```

Concurrency Management

The service implements concurrency management by using the `BlockingThreadPool` and `NonBlockingThreadPool` to limit the number of threads that can be used at any given time. This helps to prevent race conditions and other concurrency-related issues while reading and processing the reports asynchronously.

Communicating With The Aggregatable Report Accounting Service

When the `createJob` API is invoked, the snippet of code below is responsible for checking if the job is being run in debug mode using the `debugRun` variable. If the job is not being run in



debug mode, it proceeds to consume report accounting budget units (`budgetsToConsume`) from a `Aggregatable Report Accounting Service`.

The `Aggregatable Report Accounting Service` is accessed using the `privacyBudgetingServiceBridge` object, which implements the `PrivacyBudgetingServiceBridge` interface. The method `consumePrivacyBudget` is called on this object to consume the report accounting budget units. The parameters passed to the method are the list of report accounting budget units to consume and the job parameter `attributionReportTo`.

Additionally, the `missingPrivacyBudgetUnits` is checked to detect when the budget was not available for some budget units and if so, a custom exception is thrown with the message `PRIVACY_BUDGET_EXHAUSTED` and the error message `PRIVACY_BUDGET_EXHAUSTED_ERROR_MESSAGE`.

```
// Do not consume any privacy budget for debug-run jobs.
if (!debugRun) {
    try {
        // Only send request to PBS if there are units to consume budget for, the list of
        ↪ units
        // can be empty if all reports failed decryption
        ImmutableList<PrivacyBudgetUnit> budgetsToConsume =
            aggregationEngine.getPrivacyBudgetUnits();
        if (!budgetsToConsume.isEmpty()) {
            missingPrivacyBudgetUnits =
                privacyBudgetingServiceBridge.consumePrivacyBudget(
                    /* budgetsToConsume= */ budgetsToConsume,
                    /* attributionReportTo= */ job.requestInfo()
                        .getJobParameters()
                        .get(JOB_PARAM_ATTRIBUTION_REPORT_TO));
        }
    } catch (PrivacyBudgetingServiceBridgeException e) {
        throw new AggregationJobProcessException(
            PRIVACY_BUDGET_ERROR, "Exception while consuming privacy budget.", e);
    }

    if (!missingPrivacyBudgetUnits.isEmpty()) {
        throw new AggregationJobProcessException(
            PRIVACY_BUDGET_EXHAUSTED, PRIVACY_BUDGET_EXHAUSTED_ERROR_MESSAGE);
    }
}
```

Practical Privacy Implementation

Noised aggregation⁷ is a technique used to protect the privacy of users while still allowing for accurate data analysis. In this process, random noise is added to the data before it is aggregated, making it difficult to determine the exact values of individual data points. This makes it harder for ad techs to use the data to identify or track specific individuals, while still allowing meaningful insights to be drawn from the aggregated data.

The `NoisedAggregationRunner` class is responsible for running this process and adding the noise to the data before it is aggregated. The Java snippet defines an interface called `NoisedAggregationRunner`, to apply “differential private noising” to a list of `AggregatedFact` objects. The interface includes a single method called `noise()`, which takes in three arguments:

- `Iterable<AggregatedFact> aggregatedFact`: an iterable collection of `AggregatedFact` objects, containing the data that will have noise added to them

7. <https://developer.chrome.com/docs/privacy-sandbox/aggregation-service/#noise-scale>



- `boolean doThreshold`: a boolean flag indicating whether or not to apply a threshold to the data
- `Optional<Double> debugPrivacyEpsilon`: an optional value that can be used to set the level of privacy noise to be added to the data.

The `noise()` method returns an object of `NoisedAggregationResult` type, which contains the resulting list of `AggregatedFact` objects with the noise applied.

```
/** Interface to apply Differential Private Noising to {@code AggregateFact}. */
public interface NoisedAggregationRunner {

    /**
     * Applies noise and optional threshold to values on a list of {@code AggregatedFact}.
     *
     * @return list of new {@code AggregationFact} with noising applied.
     */
    NoisedAggregationResult noise(
        Iterable<AggregatedFact> aggregatedFact,
        boolean doThreshold,
        Optional<Double> debugPrivacyEpsilon);
}
```

The `NoisedAggregationRunner` class adds `noise` to the final aggregated data in order to protect the privacy of the individual data points that were used to generate the aggregate data. The noise is added in such a way that the level of privacy protection can be controlled through the use of a L1 sensitivity and epsilon, which specifies the maximum amount of privacy loss that is allowed. The noise is added to the aggregate data after it has been calculated by the `AggregationEngine`.

The actual implementation of `NoisedAggregationRunner` class below uses a the `NoiseApplier` class to apply `noise` to the `AggregatedFact` objects. The `DpNoiseApplier` class is a implementation of the `NoiseApplier` interface that uses Google's Differential Privacy library⁸ to add `noise` to the aggregate data. The `noiseMetric()` method within the `DpNoiseApplier` class is used to add Laplace noise to the aggregate data. The noise is controlled by the epsilon value, as well as the L1 sensitivity of the data. The `DpNoiseParamsFactory` class injected into the `DpNoiseApplier` class, is responsible for creating the noise parameters for the Laplace noise distribution.

```
/** {@link NoiseApplier} implementation using Google's Differential Privacy library. */
public final class DpNoiseApplier implements NoiseApplier {

    private final DpNoiseParamsFactory valueNoiseParams;

    @Inject
    DpNoiseApplier(@DpValue DpNoiseParamsFactory valueNoiseParams) {
        this.valueNoiseParams = valueNoiseParams;
    }

    @Override
    public Long noiseMetric(Long metric) {
        return noise(metric, valueNoiseParams);
    }
}
```

8. <https://github.com/google/differential-privacy>




```

private static Long noise(Long rawValue, DpNoiseParamsFactory noiseParams) {
    checkArgument(
        noiseParams.distribution().equals(Distribution.LAPLACE),
        "Only Laplace noising distribution supported. Got: " + noiseParams.distribution());

    LaplaceNoiseParams laplaceNoiseParams = noiseParams.laplace();
    return laplaceNoiseParams
        .noise()
        .addNoise(rawValue, laplaceNoiseParams.l1Sensitivity(), laplaceNoiseParams.epsilon());
}

```

Aggregatable Report Accounting Service

Design and Implementation

The Aggregatable Report Accounting Service (ARAS) is implemented in C++ starting from the `pbs_server` component, which instantiates the `PBSInstance` class. The `PBSInstance` class is subsequently responsible for initializing the major high-level components of the ARAS. In particular, it uses the `FrontEndService` class to expose API endpoints and handle incoming requests, while making use of the `AwsAuthorizer` class to perform authentication and authorization flows. The architecture of the ARAS supports multiple instances of the ARAS server itself working in tandem. Consequently, incoming requests to the `FrontEndService` are expected to come from either the Aggregation Service (see above) or other instances of the ARAS. Each instance of the ARAS runs in a trusted party external to the SCP and DP.

The ARAS is designed to perform verbose logging of each state-changing operation, in order to serve as a sort of journal. In the event of an ARAS failure, the logs are verbose enough that they may be used to recreate its exact state from before the failure occurred.

The ARAS provides budget resource management functionality using a state machine. Each instance of budget use is referred to as a “transaction”, and each transaction may be in multiple states, which are referred to as “phases”. All of the ARAS’s state-changing API calls involve either creating transactions or changing a transaction phase. The following API calls are exposed by the ARAS `FrontEndService`:

- `BeginTransaction` (HTTP POST)
- `PrepareTransaction` (HTTP POST)
- `CommitTransaction` (HTTP POST)
- `NotifyTransaction` (HTTP POST)
- `AbortTransaction` (HTTP POST)
- `EndTransaction` (HTTP POST)
- `GetTransactionStatus` (HTTP GET)
- `GetServiceStatus` (HTTP GET)

All of the POST requests accept a transaction ID (UUID) and transaction secret (string value), passed via HTTP headers. Together, these two values uniquely identify each transaction and are required in order to perform any state-changing operation on a transaction. Note that the transaction ID and secret are both chosen by the caller rather than by the ARAS itself.

The `BeginTransaction` call is designed to create a new transaction. It takes as input a JSON string containing metadata about the new transaction to create. The remaining `*Transaction` POST requests are all designed to modify the phase of a pre-existing transaction in the manner specified by the name of the call.



The remaining two GET requests are intended to retrieve basic information about the service. The `GetServiceStatus` endpoint requires no arguments, and simply returns information about the state of the service. The `GetTransactionStatus` endpoint takes a transaction ID and a transaction secret as arguments in headers, and returns information about the current state of the specified transaction.

All of the aforementioned `FrontEndService` API endpoints call into the `TransactionManager` component to carry out phase changes or retrieve data. The `TransactionManager` subsequently calls into the `TransactionEngine` component, which provides the bulk of the low-level internal implementation of each transaction operation.

Testing Methodology

NCC Group's ARAS testing consisted of a code review of the C++ components located in the `SCP\cc\pbs` directory, assisted by design documentation. At a high level, the review methodology involved analyzing the flow of user input throughout the system to identify possible undesired behavior throughout the service's implementation.

The entry points for user input were the 8 API calls exposed by the `FrontEndService` identified above. For each parameter which can be specified by the caller (including message content for each request, HTTP headers for transaction ID and transaction secret, and JSON data for `BeginTransaction`), NCC Group followed the parameter's value in the code and carefully considered what security-relevant scenarios could be triggered with it. Specific areas of focus included buffer overflows, memory leaks, concurrency issues, authentication and authorization flaws, and in-depth logic bugs.

In general, the code uses pre-established libraries, such as nlohmann's JSON library⁹, rather than using implementations written from scratch, significantly the reducing possibility of error. The code also makes use of C++11 features such as `std::atomic`¹⁰ to abstract away lower-level implementation details.

9. <https://json.nlohmann.me/>

10. <https://en.cppreference.com/w/cpp/atomic/atomic>



9 Cryptographic Design and Implementation Review

In this section, we review the design and implementation of the cryptographic elements of the Privacy Sandbox Aggregation Service. Cryptography is a security tool whose main benefit is to concentrate secrecy in a few objects (in particular cryptographic keys), allowing the abstraction of many elements of the system away from the security analysis, as it pertains to the protected data. In the Privacy Sandbox Aggregation Service, the goal is to maintain the privacy of the contents of the events generated in the client browsers. From a cryptographic point of view, the following elements are relevant:

1. Events are generated and encrypted in a browser, using a public key, whose corresponding private key is under exclusive control of the enclave code.
2. The enclave obtains the private key by recombining the two portions of the split private key. Each portion is obtained in a symmetrically-encrypted format, with a symmetric decryption key held inside an AWS Key Management Service (KMS) instance. KMS allows access to that symmetric key only to the enclave, using remote attestation, which guarantees that only enclave code with a specific configuration can use the key.
3. The encrypted events are ultimately delivered in batches to the enclave code, which proceeds to decrypt them and perform the required computation on them.
4. Before producing the output report, the enclave code consumes a reporting accounting budget, which limits the total number of requests that can be performed on some event. The budget is maintained by the Aggregatable Report Accounting Service, which is run by the coordinators.
5. The enclave uses standard TLS-based protocols to contact the coordinator(s) to retrieve the encrypted private keys, to consume reporting accounting budget, and to push output reports. Simple TLS (HTTPS) is used throughout, although the design mentions mTLS (i.e. TLS with mutual certificate-based authentication) can optionally be used for third party services such as the Aggregatable Report Accounting Service.

How encrypted events are gathered, assembled as batches, and delivered to the enclave is not relevant to cryptographic security: as long as the clients (browsers) use the right public key, with a secure encryption algorithm, and the enclave has exclusive access to the corresponding private key, event privacy will be maintained on that path. We will now study the cryptographic design of each of the elements listed above.

Public Key Encryption

The design documents for the Privacy Sandbox Aggregation Service only talk about “encryption” in a generic way. Analysis of the source code shows that the open-source Tink library¹¹ is used. Tink provides high-level cryptographic functionalities in a cross-platform and cross-language way; Privacy Sandbox uses mainly authenticated symmetric encryption (AEAD) and public-key hybrid encryption. In the case of hybrid encryption, public and private keys are exchanged as structured objects (using protobuf) that contain not only the key itself, but also the relevant metadata that document the key type and the compatible low-level cryptographic algorithms. The asymmetric key exchange mechanism, the internal key derivation function, and the symmetric encryption algorithm are specified in that metadata. The algorithm chosen for hybrid key encryption in the Privacy Sandbox Aggregation Service are ECDH (over Curve 25519), HKDF-SHA256, and CHACHA20-

11. <https://developers.google.com/tink>



Poly1305 for these three primitives, respectively. These algorithms together fulfill the properties that are expected by the Privacy Sandbox Aggregation Service:

- ECDH is a Diffie-Hellman key exchange performed over an elliptic curve. Curve 25519 is a widely used elliptic curve that offers 128-bit security. Curve 25519 is defined in RFC 7748¹².
- HKDF is a standard and secure key derivation function, adequate for expanding a secret value obtained from a Diffie-Hellman key exchange into the secrets needed for the symmetric encryption. HKDF is specified in RFC 5869¹³.
- CHACHA20-Poly1305 is an Authenticated Encryption with Additional Data (AEAD) algorithm, that ensures not only the confidentiality but also the integrity of the conveyed data. This allows recipients (here, the enclave code) to validate that the events were not altered in transit. CHACHA20-Poly1305 offers a 256-bit security level, and is defined in RFC 8439¹⁴.

The last point requires additional explanation: nothing in the system guarantees, at least cryptographically, that the source events are genuine; in fact, there is no definition of what “real” or “fake” data is, at this level. In the broader system, what prevents mass injection of fake events is a combination of browser-level and server-level mitigations, including the detection of abnormal patterns. In particular, Privacy Sandbox Aggregation Service clients are not authenticated in the cryptographic sense of the term. However, the use of an authenticated encryption algorithm (such as an AEAD) is still important because active attackers might want to alter encrypted events in an attempt to obtain information on plaintext through the behavior of the enclave code when processing the modified events (a *chosen ciphertext attack*). Use of a proper AEAD mode effectively prevents such attacks: any alteration of the ciphertext, however small, implies rejection of the whole ciphertext without revealing any information about its contents.

A crucial part of the public key encryption process is how clients obtain the public key to use. It is expected that this key will change over time; indeed, regular key rotation is planned. An active attacker might want to provide a different public key, whose private key is under the attacker’s control, so that clients encrypt events with the attacker’s key. To avoid this issue, the public key to use for asymmetric encryption of events is signed by the coordinators; clients refuse to encrypt events if they have not verified the signatures of both coordinators. Since public keys use Tink’s format, the signature covers not only the public key value, but also the metadata, which prevents algorithm confusion attacks, in which an attacker would try to make clients use the right public key with the wrong algorithm.

Recommendations

- In the interest of potential future interoperability, the documentation should specify explicitly that the encryption uses Tink’s protobuf format, with some specific primitive algorithms; such a specification would allow compatibility with non-Tink clients under special circumstances.

Retest Results

Google has addressed this recommendation in the following files:

- `java/com/google/scp/coordinator/keymanagement/keygeneration/README.md`
- `java/com/google/scp/shared/util/KeyParams.java`

12. <https://www.rfc-editor.org/rfc/rfc7748.html>

13. <https://www.rfc-editor.org/rfc/rfc5869.html>

14. <https://www.rfc-editor.org/rfc/rfc8439.html>



Private Key Generation and Key Splitting

Enclaves, by design, do not have any permanent storage ability. The private key for event decryption is stored externally in encrypted form. This encryption relies on a symmetric key which is stored in KMS and under exclusive control of the enclave (KMS is a key storage and management service, and is set-up using the attestation mechanism to ensure that it is receiving the requests from a specific enclave code).

Attestation details depend on the enclave technology; in the case of AWS Nitro Enclaves¹⁵, cryptographic hashes (“measurements”) are computed over the enclave code and then certified by the hypervisor that implements the isolation of the enclave from other systems. Access to a key stored in KMS can be limited to callers that are required to present specific cryptographic hash values. In the list of measurements offered by Nitro, the PCR0, PCR1 and PCR2 hashes are the most important here: they cover, respectively, the enclave image file, the kernel and boot filesystem, and the application; these hash values must be used conjointly to configure access to the keys. Normally, PCR0 is sufficient, since the image includes everything in the enclave, including the kernel and the application. PCR1 and PCR2 are obtained at the same time and copying them into the KMS configuration, along with PCR0, may potentially help with troubleshooting deployment failures; however, this would also enlarge the policy and possibly run into policy size constraints.

Private Key Generation: In order to prevent a malicious coordinator from keeping a copy of the private key and using it later to decrypt events outside of the enclave, the Privacy Sandbox Aggregation Service includes two coordinators, A and B, and the key pair generation is to be performed by a dedicated enclave. In this system, one coordinator controls and runs the enclave, which splits the private key into two *shares*; each of the coordinators receives a single share. Each share is encrypted using an AEAD cipher before being transmitted. The transmission of the first encrypted share to coordinator A is direct, since the coordinator runs the enclave. The second share is encrypted using a symmetric cipher, with the key taken from a KMS instance provided by Coordinator B. Attestation is used to guarantee to coordinator B that the private key is generated and split properly; that even coordinator A, who runs the enclave, cannot obtain the second share; and that the key split corresponds to the public key sent at the same time. This is done by ensuring the key provided by Coordinator B is available exclusively to an enclave running the appropriate image.

The symmetric algorithms used to encrypt the key shares are provided via KMS. None of the design documents specifies the exact cryptographic algorithms, but any secure AEAD mode could be used; CHACHA20-Poly1305 or AES/GCM is recommended.

Key Splitting is applied to the private key that the enclave will use to decrypt events. That private key is a Tink key, i.e. a protobuf-encoded structure that contains the private key itself (since ECDH is used, the private key is a randomly-chosen integer modulo the curve order) along with a copy of the corresponding public key and metadata. The key split is implemented using the following process (which is mathematically equivalent to Shamir’s secret sharing scheme with only two parties): to split the value p , a sequence of random bytes m is generated, with the same length as p ; the first share is then the bitwise XOR of p and m , and the second share is m itself. In that simplified scheme, reconstruction of p from the two shares is done by combining the two shares with a bitwise XOR. This scheme is secure as long as the mask m is generated from a cryptographically secure random source. The documentation mentions the possibility of modifying this to use multi-party Shamir secret sharing instead in the future to provide flexibility if more than two total coordinators were to be used.

15. <https://docs.aws.amazon.com/enclaves/latest/user/set-up-attestation.html>



Recommendations

- Ensure that the key generation enclave uses a secure AEAD encryption mode for symmetric encryption tasks. The exact algorithms should be documented in a design document.

Retest Results

The key generation enclave uses an AEAD encryption mode explicitly (using the `com.google.crypto.tink.Aead` interface); the mode is set when the key itself is created in AWS KMS. The “Secure Control Plane Split Key Design” document specifies that the ChaCha20+Poly1305 AEAD mechanism is to be used; this mode provides adequate security.

Aggregatable Report Accounting Service

The *report accounting budget* is a mechanism that limits the number of requests that can be performed on the collected events. Indeed, if an arbitrary number of requests can be performed on some events, then that allows reconstruction of the private data through dichotomic searches; addition of random noise to each result makes such reconstruction more expensive, but does not ultimately prevent it. To maintain the privacy of the source data, the report accounting budget is a formal quantity that is depleted whenever some processing is performed on the corresponding events; the budget is consumed by some amount that depends on the type of processing performed. No processing may be performed when the budget is insufficient. The budget can *never* be replenished.

The enclave code must enforce the budget-limited restrictions (i.e. refuse to return the requested aggregate information if there is no available budget). However, the enclave does not have any local storage capabilities as enclaves are stateless. Therefore the budget is tracked in a dedicated Aggregatable Report Accounting Service (ARAS), maintained externally to the enclave and run by the coordinators. A few salient points on this mechanism are the following:

- Report accounting budget enforcement is crucial for the overall property of event privacy; without it, an unbounded number of requests can always be performed by the entity that runs the enclave, which can leverage that ability to recover most of the private data.
- The enclave can make sure that the budget was consumed only if it could validate that it talked to the real ARAS. This property is normally ensured by the employed transport mechanism, as long as it offers authentication of the service by the enclave. A normal HTTPS call is sufficient. It is not strictly necessary that the authentication be mutual (i.e. that the ARAS authenticates the enclave) though it may help defeat denial-of-service attacks in which a fake enclave disrupts the service by consuming the budget.
- The ARAS is run as a distributed service, with both coordinators maintaining an instance of the ARAS service. The ARAS client then reaches out to both ARAS instances in parallel, and waits for acknowledgement from both before proceeding. This is done in order to preserve the atomicity of transactions, and is detailed in the “Distributed Privacy Budget Service” document, along with a description of the process used to re-synchronize the two ARAS services if they get out of sync.
- For budget usage to be properly enforced, the enclave code must ensure that the budget consumption was acknowledged by the ARAS before returning the computed aggregate results; otherwise, the enclave host could stop the enclave after production of the result but before budget depletion, thereby avoiding budget consumption. The aggregation process is implemented in `ConcurrentAggregationProcessor.java`; in this file, the call to the ARAS (via the function `consumePrivacyBudget`) indeed occurs before scheduling the



return of the result. There are a couple of small documentation mismatches regarding this point:

- In the “ADMC Multi-Cloud Secure Control Plane” document, the following description of the privacy budgeting flow is present:

- 5) The Business Logic (G)
 - a) Checks is there is Privacy Budget available to process the request
 - b) Processes the request.
 - c) Checks again if Privacy Budget is still available, and stores the result for later retrieval. Storage uses either a system-provided or a Customer-Managed Encryption Key (CMEK) for encryption at rest.

Figure 3: Privacy Budgeting Flow - ADCM Multi-Cloud Secure Control Plane

However, the aggregation process only calls the ARAS service after processing the request.

- In the file `PrivacyBudgetingServiceBridge.java`, the following comment is used to describe the behaviour of the `consumePrivacyBudget` function:

```
/**
 * Consumes the privacy budget for the given IDs and returns the budgets which cannot be
 * ↳ consumed.
 *
 * <p>If the method returns an empty list, that means the budgets were successfully
 * ↳ consumed,
 * otherwise, the first few units for which the budget could not be consumed is returned.
 *
 * @return First few privacy budgeting units for which budget consumption failed.
 */
```

This is not entirely correct: if any budgets cannot be consumed, the transaction is aborted and no budgets are consumed.

- The distributed ARAS computation occurs as a series of Begin - Prepare - Commit - Notify phases in order to preserve atomicity of transactions and ensure that the two ARAS instances remain in sync, as documented in the “Distributed Privacy Budget Service” document. During the Notify phase, the transaction can no longer be aborted and the phase will repeat until both ARAS instances have successfully processed the transaction. This computation occurs in the `executeDistributedPhase` function in `TransactionEngineImpl.java`, which will be called repeatedly until the Notify phase is complete:

```
for (PrivacyBudgetClient privacyBudgetClient : this.privacyBudgetClients) {
    ExecutionResult executionResult =
        dispatchDistributedCommand(privacyBudgetClient, transaction);
    if (executionResult.executionStatus() != ExecutionStatus.SUCCESS) {
        // Only change if the current status was false.
        if (!transaction.isCurrentPhaseFailed()) {
            transaction.setCurrentPhaseFailed(true);
            transaction.setCurrentPhaseExecutionResult(executionResult);
        }
    }
}
```

However, note that the ARAS client will wait until both ARAS implementations *simultaneously* report the transaction as “successfully” completed. Thus, if one of the ARAS implementations were to report the transaction completed successfully and then stop responding while the other ARAS implementation did not at first respond, but then report a success, this may not be recorded properly at first, and delay finalizing the aggregation computation.



Recommendations

- Ensure that the documentation regarding the Aggregatable Report Accounting Service is complete and consistent.

Retest Results

- Google updated the internal design document for the ‘ADMC Multi-Cloud Secure Control Plane’ to ensure its accuracy. In addition, Google has updated the incorrect documentation for ‘consumePrivacyBudget’ in the ‘java/com/google/aggregate/privacy/budgeting/bridge/PrivacyBudgetingServiceBridge.java’ file.
- Google augmented the loop in the `executeDistributedPhase` function to remember the last success status from a given ARAS, which avoids the stuck ARAS client situation described above.

Side-Channel Attacks

Side-channel attacks try to extract information about secret data from the observable behavior of the system that processes the data, beyond the abstract model of the computation: even if an enclave returns only an aggregate result (with added random noise), it may leak partial information about the plaintext events through various physical channels. Since AWS Nitro enclaves run in hardware hosted by AWS, it is assumed that potential attackers do not have physical access to the machines, which limits potential side channels to what can be detected and measured from the host system itself, or over a network, without any special apparatus. In practice, this means that attackers with control of the host are limited to timing measurements.

Attackers may observe not only the total time taken to process a batch, but may also gain insight about the successive steps by analyzing the *memory access pattern* of the enclave: indeed, every memory access, whether for fetching instructions to execute, or for reading or writing data, exercises the various caches that sit in the CPU between the execution units and the physical RAM. Cache misses incur extra delays (up to several hundreds of cycles), that can be detected externally. Moreover, Nitro instances have dedicated CPU cores but separate cores within the same physical CPU still share some of the cache (especially level-3 cache), so that the memory access pattern of the enclave code will impact cache layout. In particular, the enclave code might evict from cache some memory lines belonging to other (non-enclave) VMs running on the same host, and the latter VMs could detect which of their cache lines were so impacted through timing measurements when accessing them again. Such *cache attacks* leak information about the addresses at which the enclave code performs accesses, even if the byte values read from or written to these addresses do not leak.

Cryptographic implementations in modern libraries (such as Tink) are implemented in ways that avoid such information leakage, through various techniques that ensure that the memory access pattern is independent of secret data (such implementations are colloquially known as “constant-time”, though this nomenclature is slightly inadequate). This characteristic should prevent any timing attack from revealing the contents of cryptographic keys or the plaintext data while they are processed by cryptographic algorithms. However, any processing performed on the plaintext events *after* decryption may conceptually leak information about the event data, if the code performs conditional jumps that depend on the event contents or accesses memory at addresses indexed by the event value bytes. In particular, Base64 decoding and JSON parsing will almost certainly induce that kind of side channel. This implies that, from a theoretical point of view, timing attacks are possible.



Practical timing attacks on the Privacy Sandbox Aggregation Service enclave are unlikely for several reasons:

- The attacker needs to have control of a virtual machine co-hosted on the same physical hardware as the target enclave so that cache-based attacks may be set up (timing attacks over a network have been demonstrated in ideal lab conditions, but are more difficult because they require indirect cache setup and because network latencies add noise to timing measurements). AWS allocation of virtual CPUs to physical hardware is often opaque and hard to predict or assess.
- Nitro enclaves are isolated from other VMs (and from each other) through a custom hypervisor that is not documented. This hypervisor is *not* under control of the attacker. Any attacker would have to perform substantial reverse engineering of the behavior of the hypervisor, in particular, with regard to memory allocation, to work out where the target data would be stored in physical RAM. Moreover, it is likely that the custom hypervisor includes some mitigations against timing attacks, e.g. memory layout randomization. Its implementation may also be changed by AWS at any time, without notice.
- Side-channel attacks are statistical in nature, often requiring many measurements to extract the information signal. Such repeated executions may be externally detectable.

It may be argued that side-channel attacks are not a problem for Nitro-based enclaves. Other enclave technologies might be more susceptible; in particular, enclave mechanisms that are for application code only, with the (assumed potentially hostile) host running all hypervisor tasks, allow attackers to have a very precise and accurate notion of what the enclave code is currently doing, monitoring memory accesses on an almost cycle-by-cycle basis. In contrast, AWS Nitro makes the hypervisor inaccessible to attackers and should be expected to provide better isolation in the intended context.

If extra protection against side-channel attacks is desired, then one method would be to add a network call from the enclave to the coordinator(s) *before* processing the incoming batch (in particular, before even trying to decrypt the events). An attacker trying to leverage a timing attack through multiple measurements would need to send a new batch request each time (the enclave can be restarted at will, but not brought back directly to a previous snapshot without cooperation from the hypervisor); if every attempt triggers a call to the coordinators, while not being followed by the normal calls to the ARAS for report accounting budget consumption, then such attacks will be more likely to be detected. This mitigation does not prevent information leak but raises the difficulty of a timing attack at a moderate extra cost (since, in the normal course of events, a network call to the coordinator-hosted ARAS already happens).

Recommendation:

- Consider adding an initial pre-decryption call to the coordinators, so that attempts at repeated side-channel measurements on an enclave may be detected by the coordinators.

Client Response

We decided not to implement the recommendation at this stage. Timing attacks on the Privacy Sandbox Aggregation Service enclave are currently thought to be impractical and there are various protections in place to prevent secret data leakage. Specifically, the isolation provided by the Nitro-enclaves, use of cryptographic implementations and the execution time variance added by the system components makes timing attacks to gain timing information unlikely. Lastly, we believe that adding initial pre-decryption calls to the



coordinators provides limited security value. Going forward, we'll continue to assess the importance of this and other mitigations against side channel attacks.



10 Finding Details

High

Missing Public Key Integrity Check

Overall Risk	High	Finding ID	NCC-E004186-97Y
Impact	High	Component	Secure Control Plane
Exploitability	Low	Category	Cryptography
		Status	Fixed
CVSS	8.2 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N)		

Impact

The omission of the public key from the associated data during the encryption of Coordinator B's key split may lead to lack of integrity of the public key `p_public`, and enable a replay attack.

Description

During the setup of the Privacy Sandbox Aggregation Service, the public key `p_public` and corresponding private key `p_private` used for event encryption and decryption are generated within a TEE run by Coordinator A. The private key `p_private` is then split into two portions `p_1` and `p_2`, which are stored by Coordinator A and Coordinator B respectively. After being generated, the key split portion `p_2` is encrypted within the TEE using the key `k_b` provided by Coordinator B. The encrypted key split $E_{k_b}(p_2)$ and `p_public` are then sent to Coordinator B. This is documented in the "Secure Control Plane Split Key Design" document:

- 4) Coordinator A's TEE encrypts `p_2` using `k_b` and sends $E_{k_b}(p_2)$ and `p_public` to Coordinator B
 - a) (important) `p_public` is used in the associated data / shared info.
 - i) `Decrypt(payload= $E_{k_b}(p_2)$, associated_data=p_public)`
 - ii) If `p_public` is modified, decryption will fail
 - (1) This prevents a replay attack where $E_{k_b}(p_2)$ is re-used with a different public key belonging to a key generated outside a TEE.
- 5) Coordinator B decrypts `p_2` using `k_b`
 - a) Because of 2a/b above, Coordinator B verifies that `p_2` was generated inside a TEE
 - b) `p_public` is authenticated because of AEAD

In particular, the documentation notes that the public key `p_public` should be included in the associated data during encryption, in order to maintain integrity and prevent replay attacks.

The split key functionality is implemented in the `createSplitKeyBase()` function in `CreateSplitKeyTaskBase.java`:

```
KeysetHandle privateKeysetHandle = KeysetHandle.generateNew(keyTemplate);
KeysetHandle publicKeysetHandle = privateKeysetHandle.getPublicKeysetHandle();

ImmutableList<ByteString> keySplits = KeySplitUtil.xorSplit(privateKeysetHandle, 2);

EncryptionKey key =
    buildEncryptionKey(
        creationTime,
        validityInDays,
```



```
        ttlInDays,
        publicKeysetHandle,
        keyEncryptionKeyUri,
        signatureKey);
unsignedCoordinatorAKey =
    createCoordinatorAKey(keySplits.get(0), key, keyEncryptionKeyAead, keyEncryptionKeyUri);
unsignedCoordinatorBKey = createCoordinatorBKey(key);

encryptedKeySplitB = encryptPeerCoordinatorSplit(keySplits.get(1), dataKey);
```

Note that the encryption does not include the public key as part of the associated data when encrypting Coordinator B's key share. Thus, the code diverges from the design document.

Recommendation

Modify the encryption call within the `createSplitKeyBase()` function to include the `publicKeysetHandle` as part of the associated data.

Location

<https://adm-cloud-sandbox.googleusercontent.com/ncc/scp/+refs/heads/main/java/com/google/scp/coordinator/keymanagement/keygeneration/tasks/common/CreateSplitKeyTaskBase.java#128>

Retest Results

2023-04-03 – Fixed

The issue has been resolved by ensuring that the public key is used during the encryption process. The fix can be traced through the following code snippets:

1. In `CreateSplitKeyTaskBase.java` (line 161), the `encryptedKeySplitB` value is created by calling the `encryptPeerCoordinatorSplit()` function and passing the public key material as a third parameter.
2. The call then goes to `AwsCreateSplitKeyTask.java` (line 69), where the Base64-encoded encrypted key split and public key are passed to the `encryptWithDataKey()` function.
3. Finally, in `DataKeyEncryptionUtil.java` (line 52), the public key material is converted to bytes and used as additional data for AEAD encryption, which is in line with the specifications.



Low

EC2 Instance Metadata Service Version 1 In Use

Overall Risk Low

Impact High

Exploitability Low

Finding ID NCC-E004186-7DQ

Component Holistic Attacker Modeled Pentest

Category Configuration

Status Fixed

CVSS 3.3 (CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N)

Impact

The virtual machine uses IMDS v1 which has no security features to mitigate server-side request forgery attack. An attacker could gain access to the `operator-demo-env01-AggregationServiceWorkerRole` role via IMDS v1. Limited S3, DynamoDB, CloudWatch, SSM, AutoScale, and SQS permissions were assigned to this role and the attacker could use them with the access token from IMDS v1.

Description

One EC2 instance was found to have the AWS Instance Metadata Service (IMDS) Version 1 enabled. Although this version is not inherently insecure, it has been leveraged as part of a number of high profile breaches, leading to the release of Version 2, which contains additional security features targeted to mitigate the risk from potential exploitation vectors.

The Instance Metadata Service is an API used by instances to retrieve data about themselves. In Version 1, a simple request/response model is used to communicate with this API. Version 2 mitigates attacks such as server-side request forgery by implementing a session-oriented model in which requests to obtain metadata must carry a session token in a custom HTTP header. The session token can only be obtained through HTTP PUT requests that do not contain X-Forward-For headers.

For example, the following command was executed to retrieve the metadata configuration for the `aggregation-service-operator-demo-env01 (i-07c95fd65f7db8721)` instance:

```
$ aws ec2 describe-instances --instance-ids i-07c95fd65f7db8721 --query
↳ "Reservations[].Instances[]."
↳ {"HttpEndpoint:MetadataOptions.HttpEndpoint,HttpTokens:MetadataOptions.HttpTokens}"
[
  {
    "HttpEndpoint": "enabled",
    "HttpTokens": "optional"
  }
]
```

The `"HttpEndpoint": "enabled"` value indicated that instance metadata was enabled for this instance; while `"HttpTokens": "optional"` indicated that IMDSv2 was not enforced. If IMDSv2 were enabled, the `HttpTokens` parameter would be set to `required`.



By querying the IMDS service, it was possible to retrieve role credentials with limited S3, DynamoDB, CloudWatch, SSM, AutoScale, and SQS permissions:

```
curl http://169.254.169.254/latest/meta-data/iam/security-credentials/operator-demo-env01-
↳ AggregationServiceWorkerRole

{  "Code" : "Success",
   "LastUpdated" : "2023-01-19T14:30:23Z",
   "Type" : "AWS-HMAC",
   "AccessKeyId" : "ASIAWKUFLIPWSBOMB4R",
   "SecretAccessKey" : "T3c84oo0u3Nah1NP8SOG04+{REMOVED DATA}",
   "Token" : "{REMOVED DATA}",
   "Expiration" : "2023-01-19T20:42:27Z"}
```

Recommendation

The following are recommended to ensure all instances use the latest version of the IMDS:¹⁶
17 18 19

- Update SDKs, CLIs and other software using the Instance Metadata Service to IMDSv2-compatible versions
- For existing instances, it is possible to require IMDSv2 using the `modify-instance-metadata-options` command for a specific instance
- For new instances, the use of IMDSv2 can be specified using the `run-instances` command

Location

- AccountID 435145098221
- EC2 VM i-07c95fd65f7db8721

Retest Results

2023-03-31 – Fixed

The issue has been address by updating the file `operator/terraform/aws/modules/worker/main.tf` to set `metadata_options`. Enabling `metadata_options` in Terraform helps prevent SSRF attacks by restricting access to the instance metadata service from within the EC2 instance. Dynamic testing of the EC2 instance confirms that the `ssm-user` and `root` user is unable to retrieve metadata by curling `http://169.254.169.254/latest/meta-data/`.

16. AWS EC2: <https://aws.amazon.com/ec2/>

17. AWS Documentation – Configuring Instance Metadata Service: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html>

18. AWS Documentation – `modify-instance-metadata-options` command: <https://docs.aws.amazon.com/cli/latest/reference/ec2/modify-instance-metadata-options.html>

19. AWS Blog – Add defense in depth against open firewalls, reverse proxies, and SSRF vulnerabilities with enhancements to the EC2 Instance Metadata Service: <https://aws.amazon.com/blogs/security/defense-in-depth-open-firewalls-reverse-proxies-ssrf-vulnerabilities-ec2-instance-metadata-service/>



Low

Lack of Overwrite Controls in S3

Overall Risk	Low	Finding ID	NCC-E004186-9HH
Impact	Low	Component	API
Exploitability	Low	Category	Access Controls
		Status	Risk Accepted
CVSS	4.8 (CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:L)		

Impact

An authenticated attacker can overwrite data stored inside the S3 bucket where aggregated reports and outputted summary report are stored.

Description

The `createAsJob` API provides a way for operators to initiate jobs by parsing JSON data from the request body. This endpoint then initiates a job according to the parameters provided and stores the job details in a database. Additionally, it adds a message to a queue. A separate Data Processing service listens for these messages and handles the job asynchronously. The issue here is that the write location (`output_data_blob_prefix`) has the ability to overwrite existing data stored in the S3 bucket. This opens up the opportunity for an attacker that has access to the API to overwrite previous output files or even encrypted source data. Additionally, an attacker may be able to overwrite the `.tfstate` file stored in the S3 bucket. If an attacker overwrites the `.tfstate` file and the local `.tfstate` is lost, the user may be at risk of losing sensitive data stored in the `.tfstate` file.

The Aggregation Service set-up instructions do prompt the user to “Consider enabling versioning to preserve, retrieve, and restore previous versions and set appropriate policies for this bucket to prevent accidental changes and deletion.” However, relying on users to turn on this security control can be risky as it relies on the assumption that users will remember to enable it.

Given the limited usefulness of this vector from an attacker perspective and the need to know the target file name, the resulting risk rating for this finding is low.

The parameters of the `createJob` API are shown in the following JSON snippet. The `output_data_blob_prefix` parameter can be used to overwrite files if an existing file name is chosen.

```
{
  "input_data_blob_prefix": "reports.avro2",
  "input_data_bucket_name": "pentesting-demo",
  "output_data_blob_prefix": "<output name here>",
  "output_data_bucket_name": "pentesting-demo",
  "postback_url": "http://postback.com",
  "job_parameters": {
    "attribution_report_to": "foo.com"
  },
  "job_request_id": "AAAAAS"
}
```

Recommendation

- The service could automatically assign the output file name with a randomly generated string, such as a universally unique identifier (UUID) or pseudo string. This would make it difficult for an attacker to predict output file names.



-
- If the user requires the ability to assign the output file name, the service should have a check whether that output file name already exists within storage before processing the job.

Reproduction Steps

1. Generate a clean set of aggregated data via the POST `/createJob` endpoint
2. Note the `output_data_blob_prefix` name field
3. Generate a new job via POST to `/createJob` endpoint. Maintain the same `output_data_blob_prefix` name field but change the `job_request_id` field.
4. Note the new timestamp of the replaced file.

Location

<https://api-gateway/stage/v1alpha/createJob>

Retest Results

2023-03-31 – Not Fixed

According to Google's documentation, it's recommended to enable versioning on the bucket to preserve and restore previous versions, and set appropriate policies to prevent accidental changes and deletions.

Client Response

Ad tech is responsible for enabling versioning on the S3 bucket



Lack of VM Image Hardening

Overall Risk	Low	Finding ID	NCC-E004186-JPU
Impact	Medium	Component	Holistic Attacker Modeled Pentest
Exploitability	Low	Category	Configuration
		Status	Fixed
CVSS	4.5 (CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:U/C:L/I:L/A:L)		

Impact

An attacker could easily connect to other virtual machines or databases within the network, gather information about other resources and copy data out from the cloud environment without raising any alerts. Furthermore, it is possible to interact with the IMDS and gain access to a role associated with the virtual machine. Interacting with the nitro enclave is also possible.

Description

The EC2 virtual machine (VM) that was running the Nitro enclave had no hardened AMI image.^{20 21} Without hardening, the virtual machine uses default configurations that prioritize general usage instead of maximum security. The developer tool `python` can be used for executing DNS tunneling tools and data exfiltration. The networking tools `tcpdump`, `curl` and `wget` can be used for querying IMDS v1 and gaining access to the `operator-demo-env01-AggregationServiceWorkerRole`. The Package manager `yum` could be used to install tools for further attacks and to list outdated and vulnerable packages. `SSH` was also installed on the VM, which can be used for lateral movement within the VPC. It is worth noting that the VM had no public IP address assigned and had no EC2 key-pairs associated for SSH connection.

In addition, it was running without UEFI secure boot.^{22 23} This meant that it was not possible to provide verification about the state of the boot chain or to ensure that only cryptographically verified UEFI binaries were executed after the self-initialization of the firmware.

Recommendation

Create and enable UEFI secure boot for the Linux AMI image. Consider choosing a minimal image with only the necessary packages or removing any packages and binaries (including compilers, network utilities, `postfix`, `ssh`, package managers, etc) from the current image that are not necessary for running the enclave. In addition, apply kernel hardening via `sysctl` settings and file system restriction.²⁴

Location

- EC2 VM i-07c95fd65f7db8721

20. Container-Optimized OS Overview: <https://cloud.google.com/container-optimized-os/docs/concepts/features-and-benefits>

21. Security Overview: <https://cloud.google.com/container-optimized-os/docs/concepts/security>

22. How UEFI Secure Boot works: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/how-uefi-secure-boot-works.html>

23. Amazon EC2 Now Supports NitroTPM and UEFI Secure Boot: <https://aws.amazon.com/blogs/aws/amazon-ec2-now-supports-nitrotpm-and-uefi-secure-boot/>

24. Kernel Hardening Via Sysctl: <https://obscurix.github.io/security/kernel-hardening.html>



Retest Results

2023-03-31 – Fixed

Google has migrated from the default Amazon Linux image to the minimal version of Amazon Linux image 2023, resulting in a significant reduction in unnecessary packages that are not required to run the Aggregation Service.

The findings suggest that certain developer tools could pose a potential security threat. However, it should be noted that these tools also have legitimate use cases for the Aggregation Service to operate effectively. For instance, py3 (pyyaml) is used in the Nitro Enclave Allocator, curl is used in the script for managing enclaves, and yum/dnf package manager is used in creating the Amazon Machine Image (AMI).

At the time of writing, Amazon Web Services (AWS) does not support UEFI secure boot in their Amazon Linux Image 2022 and 2023. The only distribution that AWS currently offers support for UEFI is Ubuntu Linux. Google has reported that they will offer UEFI secure boot once it is supported by AWS.

Client Response

Vulnerabilities in the host would not lead to any impact on processing within the enclave, i.e. all promises made by the design are maintained. In addition, ad techs can build the code and use their own AMI; we supply an AMI only for convenience.



Docker Image with Scan Findings

Overall Risk	Low	Finding ID	NCC-E004186-R6C
Impact	High	Component	Holistic Attacker Modeled Pentest
Exploitability	Low	Category	Configuration
		Status	Fixed

CVSS 5.2 (CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:U/C:L/I:H/A:L)

Impact

Vulnerable components may allow an attacker to gain access to the VM in case of successful exploitation of these additional attack vectors.

Description

The EC2 VM (`i-07c95fd65f7db8721`) hosted the Nitro enclave service which runs the aggregation service workers. The image (`gcr.io/distroless/java@sha256:1606422cc472612cb5bcd885684b4bf87b3813246c266df473357dce5a0fb4b4`) used for Nitro enclave had vulnerabilities identified by the `Grype` and `Trivy` SAST scanning tools. Image scanning helps in identifying software vulnerabilities in container images. They use the Common Vulnerabilities and Exposures (CVEs) database and provides with a list of scan findings. Users can review the scan findings for information about the security of the container images that are being deployed.

The following `Grype` command was executed to retrieve the latest scan findings for the image:

```
grype gcr.io/distroless/
↳ java@sha256:1606422cc472612cb5bcd885684b4bf87b3813246c266df473357dce5a0fb4b4
↳ 1 x
  ✓ Vulnerability DB      [no update available]
  ✓ Pulled image
  ✓ Loaded image
  ✓ Parsed image
  ✓ Cataloged packages   [21 packages]
  ✓ Scanned image        [123 vulnerabilities]
```

The following `Trivy` command was executed to retrieve the latest scan findings for the image:

```
root@ip-10-0-100-19 tmp]# /usr/local/bin/trivy image gcr.io/distroless/
↳ java@sha256:1606422cc472612cb5bcd885684b4bf87b3813246c266df473357dce5a0fb4b4
2023-01-18T18:29:36.612Z      INFO    Need to update DB
2023-01-18T18:29:36.612Z      INFO    DB Repository: ghcr.io/aquasecurity/trivy-db
2023-01-18T18:29:36.612Z      INFO    Downloading DB...
36.07 MiB / 36.07 MiB
↳ [-----]
↳ [-----]
↳ [-----] 100.00% 11.51 MiB p/s 3.3s
2023-01-18T18:29:40.633Z      INFO    Vulnerability scanning is enabled
2023-01-18T18:29:40.633Z      INFO    Secret scanning is enabled
2023-01-18T18:29:40.633Z      INFO    If your scanning is slow, please try '--security-
↳ checks vuln' to disable secret scanning
```



```

2023-01-18T18:29:40.633Z      INFO    Please see also https://aquasecurity.github.io/trivy/
↳ v0.36/docs/secret/scanning/#recommendation for faster secret detection
2023-01-18T18:29:41.349Z      INFO    Detected OS: debian
2023-01-18T18:29:41.349Z      INFO    Detecting Debian vulnerabilities...
2023-01-18T18:29:41.355Z      INFO    Number of language-specific files: 0

gcr.io/distroless/java@sha256:1606422cc472612cb5bcd885684b4bf87b3813246c266df473357dce5a0fb4b4
↳ (debian 10.9)

Total: 128 (UNKNOWN: 3, LOW: 32, MEDIUM: 44, HIGH: 29, CRITICAL: 20)

```

As can be seen in the above output, the scanning tools identified more than 100 vulnerabilities, including 20 critical and 29 high risk findings, for this image. It was not possible to identify if any of the vulnerable packages or libraries were used by the application. Overall Risk has been set to Low rather than High, because access to the running system by the enclave was very limited. The output of the scanners can be found in the [Supplemental Data - Docker Image Scan Results](#) section.

Recommendation

Update the image and packages if any of the vulnerabilities affect the application or services.²⁵²⁶²⁷

Location

- GCP Docker registry `gcr.io/distroless/java@sha256:1606422cc472612cb5bcd885684b4bf87b3813246c266df473357dce5a0fb4b4`
- File location `aggregation-service-0.5.0/terraform/aws/control-plane-shared-libraries/WORKSPACE`

Retest Results

2023-04-03 – Partially Fixed

The Google team has provided the latest Java base image, whose hash can be found below. At the time of this writing, the new image has not yet been incorporated into the build scripts. It will be available in the next release.

As of the scan conducted using Grype, all reported vulnerabilities for the new java base image are either negligible or have been marked as ‘won’t fix’ by the package maintainers.

In addition, Google has reported that they will be implementing a patching policy to ensure that the Java base image is continuously updated, in order to mitigate any potential vulnerabilities in the image.

```

# Distroless image for running Java.
container_pull(
  name = "java_base",
  # Using SHA-256 for reproducibility.
  digest = "sha256:f88c393a67fff3f9b599eca0e90350ee27e96d3803cbc7742ec23de6a9d6dd7d",
  registry = "gcr.io",
  repository = "distroless/java11-debian11",
)

```

25. ECR Image scanning - <https://docs.aws.amazon.com/AmazonECR/latest/userguide/image-scanning.html>

26. Troubleshooting Image Scanning Issues - <https://docs.aws.amazon.com/AmazonECR/latest/userguide/image-scanning-troubleshooting.html>

27. Top 20 Dockerfile best practices - <https://sysdig.com/blog/dockerfile-best-practices/>



2023-08-03 – Fixed

At the time of writing, this finding has been successfully addressed. Google has provided an environment with a new base image that exhibits minor issues. These issues, identified through the same scanning tool, Gripe, have been classified as negligible or have been tagged as 'won't fix' by the package maintainers.

```
# Distroless image for running Java.
container_pull(
  name = "java_base",
  # Using SHA-256 for reproducibility. The tag is latest-amd64. Latest as of 2023-07-10.
  digest = "sha256:052076466984fd56979c15a9c3b7433262b0ad9aae55bc0c53d1da8ffdd829c3",
  registry = "gcr.io",
  repository = "distroless/java17-debian11",
)
```

In an effort to maintain the most updated secure images, Google has implemented a patch policy. This policy outlines strategies for implementing patches, taking into consideration factors such as the release type, release schedule, and deprecation schedule.



Container Image with Scan Findings

Overall Risk Low
Impact High
Exploitability Low

Finding ID NCC-E004186-TBC
Component Aggregatable Report
Accounting Service
Category Configuration
Status Fixed

CVSS 5.2 (CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:U/C:L/I:H/A:L)

Impact

Vulnerable components may allow an attacker to gain access to the VM in case of successful exploitation of these additional attack vectors.

Description

The container image (`gcr.io/distroless/cc@sha256:5149ac109a77868790638769902c9b088b429a82c2241a0e88547074be01713a`) used to host the Aggregatable Report Accounting Service service had vulnerabilities identified by the `Grype` and `Trivy` SAST scanning tools. Image scanning helps in identifying software vulnerabilities in container images. They use the Common Vulnerabilities and Exposures (CVEs) database and provides with a list of scan findings. Users can review the scan findings for information about the security of the container images that are being deployed.

The following `Grype` command was executed to retrieve the latest scan findings for the image:

```
# grype gcr.io/distroless/
↳ cc:latest
↳
  ✓ Vulnerability DB      [no update available]
  ✓ Pulled image
  ✓ Loaded image
  ✓ Parsed image
  ✓ Cataloged packages   [9 packages]
  ✓ Scanned image        [13 vulnerabilities]
NAME      INSTALLED      FIXED-IN  TYPE  VULNERABILITY  SEVERITY
libc6     2.31-13+deb11u5
libc6     2.31-13+deb11u5
libc6     2.31-13+deb11u5
libc6     2.31-13+deb11u5
libc6     2.31-13+deb11u5
libc6     2.31-13+deb11u5
libc6     2.31-13+deb11u5
libssl1.1 1.1.1n-0+deb11u3
libssl1.1 1.1.1n-0+deb11u3
libssl1.1 1.1.1n-0+deb11u3 (won't fix)
openssl   1.1.1n-0+deb11u3
openssl   1.1.1n-0+deb11u3
openssl   1.1.1n-0+deb11u3 (won't fix)
```



The following `Trivy` command was executed to retrieve the latest scan findings for the image:

```
# /usr/local/bin/trivy image gcr.io/distroless/cc:latest
2023-01-25T19:10:15.513Z      INFO    Need to update DB
2023-01-25T19:10:15.513Z      INFO    DB Repository: ghcr.io/aquasecurity/trivy-db
2023-01-25T19:10:15.513Z      INFO    Downloading DB...
36.21 MiB / 36.21 MiB
↳ [-----]
↳ -----] 100.00% 30.47 MiB p/s 1.4s
2023-01-25T19:10:17.335Z      INFO    Vulnerability scanning is enabled
2023-01-25T19:10:17.335Z      INFO    Secret scanning is enabled
2023-01-25T19:10:17.335Z      INFO    If your scanning is slow, please try '--security-
↳ checks vuln' to disable secret scanning
2023-01-25T19:10:17.335Z      INFO    Please see also https://aquasecurity.github.io/trivy/
↳ v0.36/docs/secret/scanning/#recommendation for faster secret detection
2023-01-25T19:10:18.625Z      INFO    Detected OS: debian
2023-01-25T19:10:18.625Z      INFO    Detecting Debian vulnerabilities...
2023-01-25T19:10:18.628Z      INFO    Number of language-specific files: 0

gcr.io/distroless/cc:latest (debian 11.6)

Total: 13 (UNKNOWN: 0, LOW: 11, MEDIUM: 2, HIGH: 0, CRITICAL: 0)
```

As can be seen in the above output, the scanning tools identified 13 vulnerabilities, including 11 low and 2 medium risk findings, for this image. It was not possible to identify if any of the vulnerable packages or libraries were used by the application. Overall Risk has been set to Low rather than High, because access to the running system is very limited. The output of the scanners can be found in the [Supplemental Data - Container Image Scan Results](#) section.

Recommendation

Update the image and packages if any of the vulnerabilities affect the application or services.

Location

- GCP Docker registry `gcr.io/distroless/java@sha256:1606422cc472612cb5bcd885684b4bf87b3813246c266df473357dce5a0fb4b4`
- File location `SCP/WORKSPACE`

Retest Results

2023-04-03 – Partially Fixed

The Google team has provided the latest Java base image, whose hash can be found below. At the time of this writing, the new image has not yet been incorporated into the build scripts. It will be available in the next release.

As of the scan conducted using Grype, all reported vulnerabilities for the new java base image are either negligible or have been marked as 'won't fix' by the package maintainers.



In addition, Google has reported that they will be implementing a patching policy to ensure that the Java base image is continuously updated, in order to mitigate any potential vulnerabilities in the image.

```
# Distroless image for running Java.
container_pull(
  name = "java_base",
  # Using SHA-256 for reproducibility.
  digest = "sha256:f88c393a67fff3f9b599eca0e90350ee27e96d3803cbc7742ec23de6a9d6dd7d",
  registry = "gcr.io",
  repository = "distroless/java11-debian11",
)
```

2023-08-03 – Fixed

At the time of writing, this finding has been successfully addressed. Google has provided an environment with a new base image that exhibits minor issues, which have been classified as negligible or have been tagged as ‘won’t fix’ by the package maintainers.

In an effort to maintain the most updated secure images, Google has implemented a patch policy. This policy outlines strategies for implementing patches, taking into consideration factors such as the release type, release schedule, and deprecation schedule.



Lambda Function Without Code Signing

Overall Risk	Informational	Finding ID	NCC-E004186-A44
Impact	Low	Component	Holistic Attacker Modeled Pentest
Exploitability	Low	Category	Data Validation
		Status	Risk Accepted
CVSS	4.1 (CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:U/C:N/I:H/A:N)		

Impact

Untrusted or unknown code can be deployed without any notification.

Description

Code signing for the Lambda Functions was not enabled and configured. When code signing is enabled, every code deployment is checked and verified to ensure that it has been signed by a trusted source. This could prevent an attacker or malicious insider from deploying unauthorized, unknown, or modified Lambda code.

Code signing events such as successful and blocked deployment can be logged with CloudTrail. If a trail is configured for these events, the administrator or security team can be notified for ongoing suspicious Lambda code deployment activities.

The following screenshot shows an example Lambda Function without code signing:

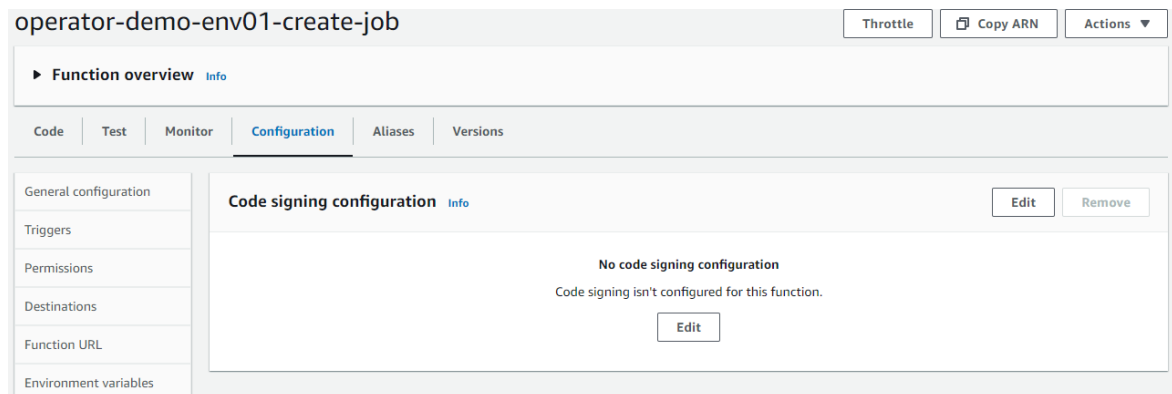


Figure 4: Code signing configuration setting

Recommendation

Configure code signing and enable logging in CloudTrail. ²⁸

Location

- AccountID 435145098221

Retest Results

2023-03-31 – Not Fixed

Client Response

Enabling and configuring code signing is the responsibility of ad tech.

28. Configuring code signing for AWS Lambda: https://docs.aws.amazon.com/lambda/latest/dg/configuration-codesigning.html?icmpid=docs_lambda_help



IAM Role Assigned with Excessive Permission

Overall Risk	Informational	Finding ID	NCC-E004186-HWM
Impact	Low	Component	Holistic Attacker Modeled Pentest
Exploitability	Low	Category	Access Controls
		Status	Risk Accepted
CVSS	2.7 (CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:L/I:N/A:N)		

Impact

An attacker that compromises the EC2 instance or `operator-demo-env01-AggregationServiceWorkerRole` role could list to more than just the Aggregation Service and/or control plane parameters.

Description

The role `operator-demo-env01-AggregationServiceWorkerRole` assigned to the running EC2 instance appears to have excessive permissions. Specifically, it has permission to list all SSM parameters in the AWS environment.

The `operator-demo-env01-AggregationServiceWorkerRole` role had the `terraform-20230105000040279800000015` permission which contained the following:

```
"Sid": "AllowSsmGetParameters",
  "Effect": "Allow",
  "Action": "ssm:GetParameters",
  "Resource": "arn:aws:ssm::parameter/*"
```

This means that any user with command line access to the machine above would be able to get information about a parameter or secret parameter from the parameter store. One way to gain access to the role is discussed in the [finding "EC2 Instance Metadata Service Version 1 In Use"](#).

Recommendation

Create security policies which follow the principle of least privilege. In this case, limit the resources that can be read from the parameter store. These policies should then be attached to the roles in use.²⁹

In particular, the role could be limited to accessing SSM parameters matching the following name patterns:

- aggregate-service
- scp-operator

Location

- AccountID 435145098221
- IAM Role operator-demo-env01-AggregationServiceWorkerRole

29. Restricting access to Systems Manager parameters using IAM policies <https://docs.aws.amazon.com/systems-manager/latest/userguide/sysman-paramstore-access.html>



Retest Results

2023-03-31 – Not Fixed

Client Response

The permission was intentionally left open to allow operators to use the parameter client to get any parameter they created themselves.



Dynamo DB Alerting Not Enabled

Overall Risk	Informational	Finding ID	NCC-E004186-HH4
Impact	Low	Component	Architecture Design Review
Exploitability	Low	Category	Auditing and Logging
		Status	Fixed
CVSS	3.3 (CVSS:3.1/AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:L/A:L)		

Impact

In the event of Dynamo database critical load, operators will not be notified before seeing impact on service performance.

Description

The Dynamo DB instances within Privacy Sandbox Aggregation Service were observed to push all logs to CloudWatch. However, no alerting policy was set.

As Cloudwatch did not have an Alert policy configured, Google would not be notified in the event of an attack or database storage saturation. This would make it difficult for Google to take remedial action before database performance began to be impacted.

Recommendation

NCC Group recommends that CloudWatch alarms should be configured for all integrated Dynamo DB instances as described in the Dynamo documentation³⁰.

Location

- Privacy Sandbox Aggregation Service

Retest Results

2023-03-31 – Fixed

The Google team resolved this issue by adding alarms within the Terraform configuration script for the Privacy Sandbox Aggregation Service and the Aggregatable Report Accounting Service to monitor potential issues. The alarms cover log thresholds, ELB error indicators, and DynamoDB table capacity alarms. When any of these alarms are triggered, notifications will be sent to the specified SNS topic.

30. Creating CloudWatch alarms to monitor DynamoDB - <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/creating-alarms.html>



11 Contact Info

The team from NCC Group has the following primary members:

- Elena Bakos Lang – Consultant
- Gage Polonsky – Project Manager
- Giacomo Pope – Consultant
- Giovanni De Ferrari – Consultant
- Huy Nguyen– Consultant
- Lydia Yao– Consultant
- Thomas Pornin – Consultant
- Tyler Colgan– Consultant
- Viktor Gazdag – Consultant

The team from Google Inc has the following primary members:

- Chanda Patel
chandapatel@google.com
- Renan Feldman
renanf@google.com
- Robert Kubis
robertkubis@google.com
- Ruchi Lohani
rlohani@google.com

Google welcomes additional feedback at the [Aggregation Service](#) GitHub repository.



12 Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Rating	Description
Critical	Implies an immediate, easily accessible threat of total compromise.
High	Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.
Medium	A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.
Low	Implies a relatively minor threat to the application.
Informational	No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

Rating	Description
High	Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.
Medium	Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.
Low	Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

Rating	Description
High	Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.



Rating	Description
Medium	Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.
Low	Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Category Name	Description
Access Controls	Related to authorization of users, and assessment of rights.
Auditing and Logging	Related to auditing of actions, or logging of problems.
Authentication	Related to the identification of users.
Configuration	Related to security configurations of servers, devices, or software.
Cryptography	Related to mathematical protections for data.
Data Exposure	Related to unintended exposure of sensitive information.
Data Validation	Related to improper reliance on the structure or values of data.
Denial of Service	Related to causing system failure.
Error Reporting	Related to the reporting of error conditions in a secure fashion.
Patching	Related to keeping software up to date.
Session Management	Related to the identification of authenticated users.
Timing	Related to race conditions, locking, or order of operations.



13 Supplemental Data - Docker Image Scan Results

Grype image scan result:

```
rype gcr.io/distroless/
↳ java@sha256:1606422cc472612cb5bcd885684b4bf87b3813246c266df473357dce5a0fb4b4
↳      1 X
  ✓ Vulnerability DB      [no update available]
  ✓ Pulled image
  ✓ Loaded image
  ✓ Parsed image
  ✓ Cataloged packages    [21 packages]
  ✓ Scanned image        [123 vulnerabilities]
NAME                INSTALLED          FIXED-IN            TYPE
↳ VULNERABILITY    SEVERITY
libc6                2.28-10            deb
↳ CVE-2010-4756    Negligible
libc6                2.28-10            deb
↳ CVE-2018-20796   Negligible
libc6                2.28-10            deb
↳ CVE-2019-1010022 Negligible
libc6                2.28-10            deb
↳ CVE-2019-1010023 Negligible
libc6                2.28-10            deb
↳ CVE-2019-1010024 Negligible
libc6                2.28-10            deb
↳ CVE-2019-1010025 Negligible
libc6                2.28-10            deb
↳ CVE-2019-9192    Negligible
libc6                2.28-10            (won't fix)        deb
↳ CVE-2020-1751    High
libc6                2.28-10            2.28-10+deb10u2    deb
↳ CVE-2016-10228   Medium
libc6                2.28-10            2.28-10+deb10u2    deb
↳ CVE-2019-19126   Low
libc6                2.28-10            2.28-10+deb10u2    deb
↳ CVE-2019-25013   Medium
libc6                2.28-10            2.28-10+deb10u2    deb
↳ CVE-2020-10029   Medium
libc6                2.28-10            2.28-10+deb10u2    deb
↳ CVE-2020-1752    High
libc6                2.28-10            2.28-10+deb10u2    deb
↳ CVE-2020-27618   Medium
libc6                2.28-10            2.28-10+deb10u2    deb
↳ CVE-2020-6096    High
libc6                2.28-10            2.28-10+deb10u2    deb
↳ CVE-2021-27645   Low
libc6                2.28-10            2.28-10+deb10u2    deb
↳ CVE-2021-3326    High
libc6                2.28-10            2.28-10+deb10u2    deb
↳ CVE-2021-33574   Critical
libc6                2.28-10            2.28-10+deb10u2    deb
↳ CVE-2021-35942   Critical
libc6                2.28-10            2.28-10+deb10u2    deb
↳ CVE-2021-3999    High
libc6                2.28-10            2.28-10+deb10u2    deb
↳ CVE-2022-23218   Critical
```



libc6	2.28-10	2.28-10+deb10u2	deb
↳ CVE-2022-23219	Critical		
libexpat1	2.2.6-2+deb10u1		deb
↳ CVE-2013-0340	Negligible		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u2	deb
↳ CVE-2021-45960	High		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u2	deb
↳ CVE-2021-46143	High		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u2	deb
↳ CVE-2022-22822	Critical		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u2	deb
↳ CVE-2022-22823	Critical		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u2	deb
↳ CVE-2022-22824	Critical		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u2	deb
↳ CVE-2022-22825	High		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u2	deb
↳ CVE-2022-22826	High		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u2	deb
↳ CVE-2022-22827	High		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u2	deb
↳ CVE-2022-23852	Critical		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u2	deb
↳ CVE-2022-23990	High		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u3	deb
↳ CVE-2022-25235	Critical		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u3	deb
↳ CVE-2022-25236	Critical		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u3	deb
↳ CVE-2022-25313	Medium		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u3	deb
↳ CVE-2022-25314	High		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u3	deb
↳ CVE-2022-25315	Critical		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u5	deb
↳ CVE-2022-40674	Critical		
libexpat1	2.2.6-2+deb10u1	2.2.6-2+deb10u6	deb
↳ CVE-2022-43680	High		
libfreetype6	2.9.1-3+deb10u2		deb
↳ CVE-2022-31782	Negligible		
libfreetype6	2.9.1-3+deb10u2	2.9.1-3+deb10u3	deb
↳ CVE-2022-27404	Critical		
libfreetype6	2.9.1-3+deb10u2	2.9.1-3+deb10u3	deb
↳ CVE-2022-27405	High		
libfreetype6	2.9.1-3+deb10u2	2.9.1-3+deb10u3	deb
↳ CVE-2022-27406	High		
libgcc1	1:8.3.0-6	(won't fix)	deb
↳ CVE-2018-12886	High		
libgcc1	1:8.3.0-6	(won't fix)	deb
↳ CVE-2019-15847	High		
libgomp1	8.3.0-6	(won't fix)	deb
↳ CVE-2018-12886	High		
libgomp1	8.3.0-6	(won't fix)	deb
↳ CVE-2019-15847	High		
libjpeg62-turbo	1:1.5.2-2+deb10u1		deb
↳ CVE-2017-15232	Negligible		
libjpeg62-turbo	1:1.5.2-2+deb10u1		deb
↳ CVE-2018-11813	Negligible		



libjpeg62-turbo	1:1.5.2-2+deb10u1		deb
↳ CVE-2020-17541	Negligible		
libjpeg62-turbo	1:1.5.2-2+deb10u1	(won't fix)	deb
↳ CVE-2020-35538	Medium		
libjpeg62-turbo	1:1.5.2-2+deb10u1	(won't fix)	deb
↳ CVE-2021-46822	Medium		
libpng16-16	1.6.36-6		deb
↳ CVE-2018-14048	Negligible		
libpng16-16	1.6.36-6		deb
↳ CVE-2018-14550	Negligible		
libpng16-16	1.6.36-6		deb
↳ CVE-2019-6129	Negligible		
libpng16-16	1.6.36-6		deb
↳ CVE-2021-4214	Negligible		
libssl1.1	1.1.1d-0+deb10u6		deb
↳ CVE-2007-6755	Negligible		
libssl1.1	1.1.1d-0+deb10u6		deb
↳ CVE-2010-0928	Negligible		
libssl1.1	1.1.1d-0+deb10u6	(won't fix)	deb
↳ CVE-2022-2097	Medium		
libssl1.1	1.1.1d-0+deb10u6	1.1.1d-0+deb10u7	deb
↳ CVE-2021-3711	Critical		
libssl1.1	1.1.1d-0+deb10u6	1.1.1d-0+deb10u7	deb
↳ CVE-2021-3712	High		
libssl1.1	1.1.1d-0+deb10u6	1.1.1d-0+deb10u8	deb
↳ CVE-2021-4160	Medium		
libssl1.1	1.1.1d-0+deb10u6	1.1.1d-0+deb10u8	deb
↳ CVE-2022-0778	High		
libssl1.1	1.1.1d-0+deb10u6	1.1.1n-0+deb10u2	deb
↳ CVE-2022-1292	Critical		
libssl1.1	1.1.1d-0+deb10u6	1.1.1n-0+deb10u3	deb
↳ CVE-2022-2068	Critical		
libstdc++6	8.3.0-6	(won't fix)	deb
↳ CVE-2018-12886	High		
libstdc++6	8.3.0-6	(won't fix)	deb
↳ CVE-2019-15847	High		
libuuid1	2.33.1-0.1		deb
↳ CVE-2022-0563	Negligible		
libuuid1	2.33.1-0.1	(won't fix)	deb
↳ CVE-2021-37600	Low		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	(won't fix)	deb
↳ CVE-2022-21619	Low		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	(won't fix)	deb
↳ CVE-2022-21624	Low		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	(won't fix)	deb
↳ CVE-2022-21626	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	(won't fix)	deb
↳ CVE-2022-21628	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	(won't fix)	deb
↳ CVE-2022-39399	Low		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.11+9-1~deb10u1	deb
↳ CVE-2021-2163	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.12+7-2~deb10u1	deb
↳ CVE-2021-2341	Low		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.12+7-2~deb10u1	deb
↳ CVE-2021-2369	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.12+7-2~deb10u1	deb
↳ CVE-2021-2388	High		



openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.13+8-1~deb10u1	deb
↳ CVE-2021-35550	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.13+8-1~deb10u1	deb
↳ CVE-2021-35556	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.13+8-1~deb10u1	deb
↳ CVE-2021-35559	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.13+8-1~deb10u1	deb
↳ CVE-2021-35561	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.13+8-1~deb10u1	deb
↳ CVE-2021-35564	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.13+8-1~deb10u1	deb
↳ CVE-2021-35565	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.13+8-1~deb10u1	deb
↳ CVE-2021-35567	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.13+8-1~deb10u1	deb
↳ CVE-2021-35578	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.13+8-1~deb10u1	deb
↳ CVE-2021-35586	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.13+8-1~deb10u1	deb
↳ CVE-2021-35603	Low		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21248	Low		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21277	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21282	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21283	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21291	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21293	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21294	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21296	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21299	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21305	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21340	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21341	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21360	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21365	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.14+9-1~deb10u1	deb
↳ CVE-2022-21366	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.15+10-1~deb10u1	deb
↳ CVE-2022-21426	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.15+10-1~deb10u1	deb
↳ CVE-2022-21434	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.15+10-1~deb10u1	deb
↳ CVE-2022-21443	Low		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.15+10-1~deb10u1	deb
↳ CVE-2022-21476	High		



openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.15+10-1~deb10u1	deb
↳ CVE-2022-21496	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.16+8-1~deb10u1	deb
↳ CVE-2022-21540	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.16+8-1~deb10u1	deb
↳ CVE-2022-21541	Medium		
openjdk-11-jre-headless	11.0.9.1+1-1~deb10u2	11.0.16+8-1~deb10u1	deb
↳ CVE-2022-34169	High		
openssl	1.1.1d-0+deb10u6		deb
↳ CVE-2007-6755	Negligible		
openssl	1.1.1d-0+deb10u6		deb
↳ CVE-2010-0928	Negligible		
openssl	1.1.1d-0+deb10u6	(won't fix)	deb
↳ CVE-2022-2097	Medium		
openssl	1.1.1d-0+deb10u6	1.1.1d-0+deb10u7	deb
↳ CVE-2021-3711	Critical		
openssl	1.1.1d-0+deb10u6	1.1.1d-0+deb10u7	deb
↳ CVE-2021-3712	High		
openssl	1.1.1d-0+deb10u6	1.1.1d-0+deb10u8	deb
↳ CVE-2021-4160	Medium		
openssl	1.1.1d-0+deb10u6	1.1.1d-0+deb10u8	deb
↳ CVE-2022-0778	High		
openssl	1.1.1d-0+deb10u6	1.1.1n-0+deb10u2	deb
↳ CVE-2022-1292	Critical		
openssl	1.1.1d-0+deb10u6	1.1.1n-0+deb10u3	deb
↳ CVE-2022-2068	Critical		
zlib1g	1:1.2.11.dfsg-1	1:1.2.11.dfsg-1+deb10u1	deb
↳ CVE-2018-25032	High		
zlib1g	1:1.2.11.dfsg-1	1:1.2.11.dfsg-1+deb10u2	deb
↳ CVE-2022-37434	Critical		

Trivy image scan result:

```

/usr/local/bin/trivy image gcr.io/distroless/
↳ java@sha256:1606422cc472612cb5bcd885684b4bf87b3813246c266df473357dce5a0fb4b4
2023-01-18T18:29:36.612Z      INFO    Need to update DB
2023-01-18T18:29:36.612Z      INFO    DB Repository: gcr.io/aquasecurity/trivy-db
2023-01-18T18:29:36.612Z      INFO    Downloading DB...
36.07 MiB / 36.07 MiB
↳ [-----]
↳ -----] 100.00% 11.51 MiB p/s 3.3s
2023-01-18T18:29:40.633Z      INFO    Vulnerability scanning is enabled
2023-01-18T18:29:40.633Z      INFO    Secret scanning is enabled
2023-01-18T18:29:40.633Z      INFO    If your scanning is slow, please try '--security-
↳ checks vuln' to disable secret scanning
2023-01-18T18:29:40.633Z      INFO    Please see also https://aquasecurity.github.io/trivy/
↳ v0.36/docs/secret/scanning/#recommendation for faster secret detection
2023-01-18T18:29:41.349Z      INFO    Detected OS: debian
2023-01-18T18:29:41.349Z      INFO    Detecting Debian vulnerabilities...
2023-01-18T18:29:41.355Z      INFO    Number of language-specific files: 0

gcr.io/distroless/java@sha256:1606422cc472612cb5bcd885684b4bf87b3813246c266df473357dce5a0fb4b4
↳ (debian 10.9)

Total: 128 (UNKNOWN: 3, LOW: 32, MEDIUM: 44, HIGH: 29, CRITICAL: 20)

```



Library	Vulnerability	Severity	Installed Version	Fixed
Version	Title			
libc6	CVE-2021-33574	CRITICAL	2.28-10	
↳ 2.28-10+deb10u2	glibc: mq_notify does not handle separately allocated thread attributes			
↳	https://avd.aquasec.com/nvd/cve-2021-33574			
↳				
↳	CVE-2021-35942			
↳	glibc: Arbitrary read in wordexp()			
↳	https://avd.aquasec.com/nvd/cve-2021-35942			
↳				
↳	CVE-2022-23218			
↳	glibc: Stack-based buffer overflow in svcunix_create via long pathnames			
↳	https://avd.aquasec.com/nvd/cve-2022-23218			
↳				
↳	CVE-2022-23219			
↳	glibc: Stack-based buffer overflow in sunrpc clnt_create via a long pathname			
↳	https://avd.aquasec.com/nvd/cve-2022-23219			
↳				
↳	CVE-2020-1751	HIGH		
↳	glibc: array overflow in backtrace functions for powerpc			
↳	https://avd.aquasec.com/nvd/cve-2020-1751			
↳				
↳	CVE-2020-1752			
↳ 2.28-10+deb10u2	glibc: use-after-free in glob() function when expanding ~user			
↳	https://avd.aquasec.com/nvd/cve-2020-1752			
↳				
↳	CVE-2020-6096			
↳	glibc: signed comparison vulnerability in the ARMv7 memcpy function			
↳	https://avd.aquasec.com/nvd/cve-2020-6096			
↳				
↳				



↳	CVE-2021-3326		
↳	glibc: Assertion failure in ISO-2022-JP-3 gconv module		
↳	related to combining characters		
↳	https://avd.aquasec.com/nvd/cve-2021-3326		
↳	-----		
↳	CVE-2021-3999		
↳	glibc: Off-by-one buffer overflow/underflow in getcwd()		
↳	https://avd.aquasec.com/nvd/cve-2021-3999		
↳	-----		
↳	CVE-2016-10228	MEDIUM	
↳	glibc: iconv program can hang when invoked with the -c		
↳	option		
↳	https://avd.aquasec.com/nvd/cve-2016-10228		
↳	-----		
↳	CVE-2019-25013		
↳	glibc: buffer over-read in iconv when processing invalid		
↳	multi-byte input sequences in...		
↳	https://avd.aquasec.com/nvd/cve-2019-25013		
↳	-----		
↳	CVE-2020-10029		
↳	glibc: stack corruption from crafted input in cosl, sinl,		
↳	sincosl, and tanl...		
↳	https://avd.aquasec.com/nvd/cve-2020-10029		
↳	-----		
↳	CVE-2020-27618		
↳	glibc: iconv when processing invalid multi-byte input		
↳	sequences fails to advance the...		
↳	https://avd.aquasec.com/nvd/cve-2020-27618		
↳	-----		
↳	CVE-2010-4756	LOW	
↳	glibc: glob implementation can cause excessive CPU and		
↳	memory consumption due to...		
↳	https://avd.aquasec.com/nvd/cve-2010-4756		
↳	-----		
↳	CVE-2018-20796		
↳	glibc: uncontrolled recursion in function		
↳	check_dst_limits_calc_pos_1 in posix/regexec.c		



		https://avd.aquasec.com/nvd/cve-2022-22822	
		CVE-2022-22823	
		expat: Integer overflow in build_model in xmlparse.c	
		https://avd.aquasec.com/nvd/cve-2022-22823	
		CVE-2022-22824	
		expat: Integer overflow in defineAttribute in xmlparse.c	
		https://avd.aquasec.com/nvd/cve-2022-22824	
		CVE-2022-23852	
		expat: Integer overflow in function XML_GetBuffer	
		https://avd.aquasec.com/nvd/cve-2022-23852	
2.2.6-2+deb10u3		CVE-2022-25235	
		expat: Malformed 2- and 3-byte UTF-8 sequences can lead to arbitrary code...	
		https://avd.aquasec.com/nvd/cve-2022-25235	
		CVE-2022-25236	
		expat: Namespace-separator characters in "xmlns[:prefix]" attribute values can lead to arbitrary code...	
		https://avd.aquasec.com/nvd/cve-2022-25236	
		CVE-2022-25315	
		expat: Integer overflow in storeRawNames()	
		https://avd.aquasec.com/nvd/cve-2022-25315	
2.2.6-2+deb10u5		CVE-2022-40674	
		expat: a use-after-free in the doContent function in xmlparse.c	
		https://avd.aquasec.com/nvd/cve-2022-40674	
2.2.6-2+deb10u2		CVE-2021-45960	HIGH
		expat: Large number of prefixed XML attributes on a single tag can...	
		https://avd.aquasec.com/nvd/cve-2021-45960	



	CVE-2021-46143		
	expat: Integer overflow in doProlog in xmlparse.c		
	https://avd.aquasec.com/nvd/cve-2021-46143		
	CVE-2022-22825		
	expat: Integer overflow in lookup in xmlparse.c		
	https://avd.aquasec.com/nvd/cve-2022-22825		
	CVE-2022-22826		
	expat: Integer overflow in nextScaffoldPart in xmlparse.c		
	https://avd.aquasec.com/nvd/cve-2022-22826		
	CVE-2022-22827		
	expat: Integer overflow in storeAtts in xmlparse.c		
	https://avd.aquasec.com/nvd/cve-2022-22827		
	CVE-2022-23990		
	expat: integer overflow in the doProlog function		
	https://avd.aquasec.com/nvd/cve-2022-23990		
↳ 2.2.6-2+deb10u3	CVE-2022-25314		
	expat: Integer overflow in copyString()		
	https://avd.aquasec.com/nvd/cve-2022-25314		
↳ 2.2.6-2+deb10u6	CVE-2022-43680		
	expat: use-after free caused by overeager destruction of a		
	shared DTD in...		
	https://avd.aquasec.com/nvd/cve-2022-43680		
↳ 2.2.6-2+deb10u3	CVE-2022-25313	MEDIUM	
	expat: Stack exhaustion in doctype parsing		
	https://avd.aquasec.com/nvd/cve-2022-25313		
	CVE-2013-0340	LOW	
	expat: internal entity expansion		
	https://avd.aquasec.com/nvd/cve-2013-0340		



↳ 2.2.6-2+deb10u4	DSA-5085-2	UNKNOWN		
	expat - regression update			
↳				
libfreetype6	CVE-2022-27404	CRITICAL	2.9.1-3+deb10u2	
↳ 2.9.1-3+deb10u3	FreeType: Buffer overflow in sfnt_init_face			
↳	https://avd.aquasec.com/nvd/cve-2022-27404			
↳	↳			
↳	CVE-2022-27405	HIGH		
↳	FreeType: Segmentation violation via FNT_Size_Request			
↳	https://avd.aquasec.com/nvd/cve-2022-27405			
↳	↳			
↳	CVE-2022-27406			
↳	FreeType: Segmentation violation via FT_Request_Size			
↳	https://avd.aquasec.com/nvd/cve-2022-27406			
↳	↳			
↳	CVE-2022-31782	LOW		
↳	ftbench.c in FreeType Demo Programs through 2.12.1 has a			
↳	heap-based bu			
↳	https://avd.aquasec.com/nvd/cve-2022-31782			
↳				
libgcc1	CVE-2018-12886	HIGH	8.3.0-6	
↳	gcc: spilling of stack protection address in cfgexpand.c and			
↳	function.c leads to...			
↳	https://avd.aquasec.com/nvd/cve-2018-12886			
↳	↳			
↳	CVE-2019-15847			
↳	gcc: POWER9 "DARN" RNG intrinsic produces repeated output			
↳	https://avd.aquasec.com/nvd/cve-2019-15847			
↳				
libgomp1	CVE-2018-12886			
↳	gcc: spilling of stack protection address in cfgexpand.c and			
↳	function.c leads to...			
↳	https://avd.aquasec.com/nvd/cve-2018-12886			
↳	↳			
↳	CVE-2019-15847			
↳	gcc: POWER9 "DARN" RNG intrinsic produces repeated output			
↳	https://avd.aquasec.com/nvd/cve-2019-15847			
↳				



libjpeg62-turbo	CVE-2020-35538	MEDIUM	1:1.5.2-2+deb10u1
	libjpeg-turbo: Null pointer dereference in		
	jcopy_sample_rows() function		
	https://avd.aquasec.com/nvd/cve-2020-35538		
<hr/>			
	CVE-2021-46822		
	libjpeg-turbo: heap buffer overflow in get_word_rgb_row() in		
	rdppm.c		
	https://avd.aquasec.com/nvd/cve-2021-46822		
<hr/>			
	CVE-2017-15232	LOW	
	libjpeg-turbo: NULL pointer dereference in jdpostct.c and		
	jquant1.c		
	https://avd.aquasec.com/nvd/cve-2017-15232		
<hr/>			
	CVE-2018-11813		
	libjpeg: "cjpeg" utility large loop because read_pixel in		
	rdtarga.c mishandles EOF		
	https://avd.aquasec.com/nvd/cve-2018-11813		
<hr/>			
	CVE-2020-17541		
	libjpeg-turbo: Stack-based buffer overflow in the		
	"transform" component		
	https://avd.aquasec.com/nvd/cve-2020-17541		
<hr/>			
libpng16-16	CVE-2018-14048		1.6.36-6
	libpng: Segmentation fault in png.c:png_free_data function		
	causing denial of service		
	https://avd.aquasec.com/nvd/cve-2018-14048		
<hr/>			
	CVE-2018-14550		
	libpng: Stack-based buffer overflow in		
	contrib/pngminus/pnm2png.c:get_token() potentially leading		
	to arbitrary code execution...		
	https://avd.aquasec.com/nvd/cve-2018-14550		



	CVE-2019-6129		
	libpng: memory leak of png_info struct in pngcp.c		
	https://avd.aquasec.com/nvd/cve-2019-6129		
	CVE-2021-4214		
	libpng: hardcoded value leads to heap-overflow		
	https://avd.aquasec.com/nvd/cve-2021-4214		
libssl1.1	CVE-2021-3711	CRITICAL	1.1.1d-0+deb10u6
1.1.1d-0+deb10u7	openssl: SM2 Decryption Buffer Overflow		
	https://avd.aquasec.com/nvd/cve-2021-3711		
	CVE-2022-1292		
1.1.1n-0+deb10u2	openssl: c_rehash script allows command injection		
	https://avd.aquasec.com/nvd/cve-2022-1292		
	CVE-2022-2068		
1.1.1n-0+deb10u3	openssl: the c_rehash script allows command injection		
	https://avd.aquasec.com/nvd/cve-2022-2068		
	CVE-2021-3712	HIGH	
1.1.1d-0+deb10u7	openssl: Read buffer overruns processing ASN.1 strings		
	https://avd.aquasec.com/nvd/cve-2021-3712		
	CVE-2022-0778		
1.1.1d-0+deb10u8	openssl: Infinite loop in BN_mod_sqrt() reachable when parsing certificates		
	https://avd.aquasec.com/nvd/cve-2022-0778		
	CVE-2021-4160	MEDIUM	
	openssl: Carry propagation bug in the MIPS32 and MIPS64 squaring procedure		
	https://avd.aquasec.com/nvd/cve-2021-4160		
	CVE-2022-2097		
	openssl: AES OCB fails to encrypt some bytes		
	https://avd.aquasec.com/nvd/cve-2022-2097		



	CVE-2007-6755	LOW	
	Dual_EC_DRBG: weak pseudo random number generator		
	https://avd.aquasec.com/nvd/cve-2007-6755		
	CVE-2010-0928		
	openssl: RSA authentication weakness		
	https://avd.aquasec.com/nvd/cve-2010-0928		
libstdc++6	CVE-2018-12886	HIGH	8.3.0-6
	gcc: spilling of stack protection address in cfgexpand.c and		
	function.c leads to...		
	https://avd.aquasec.com/nvd/cve-2018-12886		
	CVE-2019-15847		
	gcc: POWER9 "DARN" RNG intrinsic produces repeated output		
	https://avd.aquasec.com/nvd/cve-2019-15847		
libuuid1	CVE-2021-37600	LOW	2.33.1-0.1
	util-linux: integer overflow can lead to buffer overflow in		
	get_sem_elements() in sys-utils/ipcutils.c...		
	https://avd.aquasec.com/nvd/cve-2021-37600		
	CVE-2022-0563		
	util-linux: partial disclosure of arbitrary files in chfn		
	and chsh when compiled...		
	https://avd.aquasec.com/nvd/cve-2022-0563		
openjdk-11-jre-headless 11.0.12+7-2~deb10u1	CVE-2021-2388	HIGH	11.0.9.1+1-1~deb10u2
	OpenJDK: Incorrect comparison during range check elimination		
	(Hotspot, 8264066)		
	https://avd.aquasec.com/nvd/cve-2021-2388		
11.0.15+10-1~deb10u1	CVE-2022-21476		
	OpenJDK: Defective secure validation in Apache Santuario		
	(Libraries, 8278008)		
	https://avd.aquasec.com/nvd/cve-2022-21476		



↳ 11.0.16+8-1~deb10u1	CVE-2022-34169		
	OpenJDK: integer truncation issue in Xalan-J (JAXP, 8285407)		
	https://avd.aquasec.com/nvd/cve-2022-34169		
↳ 11.0.11+9-1~deb10u1	CVE-2021-2163	MEDIUM	
	OpenJDK: Incomplete enforcement of JAR signing disabled		
	algorithms (Libraries, 8249906)		
	https://avd.aquasec.com/nvd/cve-2021-2163		
↳ 11.0.12+7-2~deb10u1	CVE-2021-2369		
	OpenJDK: Incorrect verification of JAR files with multiple		
	MANIFEST.MF files (Library, 8260967)...		
	https://avd.aquasec.com/nvd/cve-2021-2369		
↳ 11.0.13+8-1~deb10u1	CVE-2021-35550		
	OpenJDK: Weak ciphers preferred over stronger ones for TLS		
	(JSSE, 8264210)		
	https://avd.aquasec.com/nvd/cve-2021-35550		
	CVE-2021-35556		
	OpenJDK: Excessive memory allocation in RTFParser (Swing,		
	8265167)		
	https://avd.aquasec.com/nvd/cve-2021-35556		
	CVE-2021-35559		
	OpenJDK: Excessive memory allocation in RTFReader (Swing,		
	8265580)		
	https://avd.aquasec.com/nvd/cve-2021-35559		
	CVE-2021-35561		
	OpenJDK: Excessive memory allocation in HashMap and HashSet		
	(Utility, 8266097)		
	https://avd.aquasec.com/nvd/cve-2021-35561		
	CVE-2021-35564		
	OpenJDK: Certificates with end dates too far in the future		



		can corrupt...	
		https://avd.aquasec.com/nvd/cve-2021-35564	
		CVE-2021-35565	
		OpenJDK: Loop in HttpsServer triggered during TLS session	
		close (JSSE, 8254967)	
		https://avd.aquasec.com/nvd/cve-2021-35565	
		CVE-2021-35567	
		OpenJDK: Incorrect principal selection when using Kerberos	
		Constrained Delegation (Libraries, 8266689)	
		https://avd.aquasec.com/nvd/cve-2021-35567	
		CVE-2021-35578	
		OpenJDK: Unexpected exception raised during TLS handshake	
		(JSSE, 8267729)	
		https://avd.aquasec.com/nvd/cve-2021-35578	
		CVE-2021-35586	
		OpenJDK: Excessive memory allocation in BMPImageReader	
		(ImageIO, 8267735)	
		https://avd.aquasec.com/nvd/cve-2021-35586	
11.0.14+9-1~deb10u1		CVE-2022-21277	
		OpenJDK: Incorrect reading of TIFF files in	
		TIFFNullDecompressor (ImageIO, 8270952)	
		https://avd.aquasec.com/nvd/cve-2022-21277	
		CVE-2022-21282	
		OpenJDK: Insufficient URI checks in the XSLT TransformerImpl	
		(JAXP, 8270492)	
		https://avd.aquasec.com/nvd/cve-2022-21282	
		CVE-2022-21283	
		OpenJDK: Unexpected exception thrown in regex Pattern	
		(Libraries, 8268813)	



↳		https://avd.aquasec.com/nvd/cve-2022-21283
↳		
↳	CVE-2022-21291	
↳		OpenJDK: Incorrect marking of writeable fields (Hotspot, 8270386)
↳		https://avd.aquasec.com/nvd/cve-2022-21291
↳		
↳	CVE-2022-21293	
↳		OpenJDK: Incomplete checks of StringBuffer and StringBuilder during deserialization (Libraries, 8270392)
↳		https://avd.aquasec.com/nvd/cve-2022-21293
↳		
↳	CVE-2022-21294	
↳		OpenJDK: Incorrect IdentityHashMap size checks during deserialization (Libraries, 8270416)
↳		https://avd.aquasec.com/nvd/cve-2022-21294
↳		
↳	CVE-2022-21296	
↳		OpenJDK: Incorrect access checks in XMLEntityManager (JAXP, 8270498)
↳		https://avd.aquasec.com/nvd/cve-2022-21296
↳		
↳	CVE-2022-21299	
↳		OpenJDK: Infinite loop related to incorrect handling of newlines in XMLEntityScanner (JAXP,...)
↳		https://avd.aquasec.com/nvd/cve-2022-21299
↳		
↳	CVE-2022-21305	
↳		OpenJDK: Array indexing issues in LIRGenerator (Hotspot, 8272014)
↳		https://avd.aquasec.com/nvd/cve-2022-21305
↳		
↳	CVE-2022-21340	
↳		OpenJDK: Excessive resource use when reading JAR manifest attributes (Libraries, 8272026)
↳		https://avd.aquasec.com/nvd/cve-2022-21340



	CVE-2022-21341	
	OpenJDK: Insufficient checks when deserializing exceptions	
	in ObjectInputStream (Serialization, 8272236)	
	https://avd.aquasec.com/nvd/cve-2022-21341	
	CVE-2022-21360	
	OpenJDK: Excessive memory allocation in BMPImageReader	
	(ImageIO, 8273756)	
	https://avd.aquasec.com/nvd/cve-2022-21360	
	CVE-2022-21365	
	OpenJDK: Integer overflow in BMPImageReader (ImageIO,	
	8273838)	
	https://avd.aquasec.com/nvd/cve-2022-21365	
	CVE-2022-21366	
	OpenJDK: Excessive memory allocation in TIFF*Decompressor	
	(ImageIO, 8274096)	
	https://avd.aquasec.com/nvd/cve-2022-21366	
11.0.15+10-1~deb10u1	CVE-2022-21426	
	OpenJDK: Unbounded memory allocation when compiling crafted	
	XPath expressions (JAXP, 8270504)	
	https://avd.aquasec.com/nvd/cve-2022-21426	
	CVE-2022-21434	
	OpenJDK: Improper object-to-string conversion in	
	AnnotationInvocationHandler (Libraries, 8277672)	
	https://avd.aquasec.com/nvd/cve-2022-21434	
	CVE-2022-21496	
	OpenJDK: URI parsing inconsistencies (JNDI, 8278972)	
	https://avd.aquasec.com/nvd/cve-2022-21496	
11.0.16+8-1~deb10u1	CVE-2022-21540	
	OpenJDK: class compilation issue (Hotspot, 8281859)	



				https://avd.aquasec.com/nvd/cve-2022-21540
				CVE-2022-21541
				OpenJDK: improper restriction of MethodHandle.invokeBasic()
				(Hotspot, 8281866)
				https://avd.aquasec.com/nvd/cve-2022-21541
				CVE-2022-21626
				OpenJDK: excessive memory allocation in X.509 certificate
				parsing (Security, 8286533)
				https://avd.aquasec.com/nvd/cve-2022-21626
				CVE-2022-21628
				OpenJDK: HttpServer no connection count limit (Lightweight
				HTTP Server, 8286918)
				https://avd.aquasec.com/nvd/cve-2022-21628
				CVE-2023-21835
				Vulnerability in the Oracle Java SE, Oracle GraalVM
				Enterprise Edition ...
				https://avd.aquasec.com/nvd/cve-2023-21835
11.0.12+7-2~deb10u1		LOW		CVE-2021-2341
				OpenJDK: FTP PASV command response can cause FtpClient to
				connect to arbitrary...
				https://avd.aquasec.com/nvd/cve-2021-2341
11.0.13+8-1~deb10u1				CVE-2021-35603
				OpenJDK: Non-constant comparison during TLS handshakes
				(JSSE, 8269618)
				https://avd.aquasec.com/nvd/cve-2021-35603
11.0.14+9-1~deb10u1				CVE-2022-21248
				OpenJDK: Incomplete deserialization class filtering in
				ObjectInputStream (Serialization, 8264934)
				https://avd.aquasec.com/nvd/cve-2022-21248



11.0.15+10-1~deb10u1	CVE-2022-21443			
	OpenJDK: Missing check for negative ObjectIdentifier			
				(Libraries, 8275151)
				https://avd.aquasec.com/nvd/cve-2022-21443
	CVE-2022-21619			
	OpenJDK: improper handling of long NTLM client hostnames			
				(Security, 8286526)
				https://avd.aquasec.com/nvd/cve-2022-21619
	CVE-2022-21624			
	OpenJDK: insufficient randomization of JNDI DNS port numbers			
				(JNDI, 8286910)
				https://avd.aquasec.com/nvd/cve-2022-21624
	CVE-2022-39399			
	OpenJDK: missing SNI caching in HTTP/2 (Networking, 8289366)			
				https://avd.aquasec.com/nvd/cve-2022-39399
	CVE-2023-21843			
	Vulnerability in the Oracle Java SE, Oracle GraalVM			
				Enterprise Edition ...
				https://avd.aquasec.com/nvd/cve-2023-21843
openssl	CVE-2021-3711	CRITICAL	1.1.1d-0+deb10u6	
1.1.1d-0+deb10u7	openssl: SM2 Decryption Buffer Overflow			
				https://avd.aquasec.com/nvd/cve-2021-3711
	CVE-2022-1292			
1.1.1n-0+deb10u2	openssl: c_rehash script allows command injection			
				https://avd.aquasec.com/nvd/cve-2022-1292
	CVE-2022-2068			
1.1.1n-0+deb10u3	openssl: the c_rehash script allows command injection			
				https://avd.aquasec.com/nvd/cve-2022-2068



↳ 1.1.1d-0+deb10u7	CVE-2021-3712	HIGH	
	openssl: Read buffer overruns processing ASN.1 strings		
↳	https://avd.aquasec.com/nvd/cve-2021-3712		

↳ 1.1.1d-0+deb10u8	CVE-2022-0778		
	openssl: Infinite loop in BN_mod_sqrt() reachable when parsing certificates		
↳	https://avd.aquasec.com/nvd/cve-2022-0778		

↳	CVE-2021-4160	MEDIUM	
	openssl: Carry propagation bug in the MIPS32 and MIPS64 squaring procedure		
↳	https://avd.aquasec.com/nvd/cve-2021-4160		

↳	CVE-2022-2097		
	openssl: AES OCB fails to encrypt some bytes		
↳	https://avd.aquasec.com/nvd/cve-2022-2097		

↳	CVE-2007-6755	LOW	
	Dual_EC_DRBG: weak pseudo random number generator		
↳	https://avd.aquasec.com/nvd/cve-2007-6755		

↳	CVE-2010-0928		
	openssl: RSA authentication weakness		
↳	https://avd.aquasec.com/nvd/cve-2010-0928		

tzdata	DLA-3134-1	UNKNOWN	2021a-0+deb10u1
↳ 2021a-0+deb10u7	tzdata - new timezone database		

↳ 2021a-0+deb10u8	DLA-3161-1		

zlib1g	CVE-2022-37434	CRITICAL	1:1.2.11.dfsg-1
↳ 1:1.2.11.dfsg-1+deb10u2	zlib: heap-based buffer over-read and overflow in inflate() in inflate.c via a...		
↳	https://avd.aquasec.com/nvd/cve-2022-37434		



↳ 1:1.2.11.dfsg-1+deb10u1	CVE-2018-25032	HIGH	
	zlib: A flaw found in zlib when compressing (not		
	decompressing) certain inputs...		
	https://avd.aquasec.com/nvd/cve-2018-25032		



libc6	CVE-2010-4756	LOW	2.31-13+deb11u5		glibc: glob
↳ implementation can cause excessive CPU and					
					memory
↳ consumption due to...					
↳ avd.aquasec.com/nvd/cve-2010-4756					

	CVE-2018-20796				glibc:
↳ uncontrolled recursion in function					
↳ check_dst_limits_calc_pos_1 in posix/regexec.c					
↳ avd.aquasec.com/nvd/cve-2018-20796					

	CVE-2019-1010022				glibc: stack
↳ guard protection bypass					
↳ avd.aquasec.com/nvd/cve-2019-1010022					

	CVE-2019-1010023				glibc: running
↳ ldd on malicious ELF leads to code execution					
					because
↳ of...					
↳ avd.aquasec.com/nvd/cve-2019-1010023					

	CVE-2019-1010024				glibc: ASLR
↳ bypass using cache of thread stack and heap					
↳ avd.aquasec.com/nvd/cve-2019-1010024					

	CVE-2019-1010025				glibc:
↳ information disclosure of heap addresses of					
↳ pthread_created thread					
↳ avd.aquasec.com/nvd/cve-2019-1010025					

	CVE-2019-9192				glibc:
↳ uncontrolled recursion in function					
↳ check_dst_limits_calc_pos_1 in posix/regexec.c					
↳ avd.aquasec.com/nvd/cve-2019-9192					

libssl1.1	CVE-2022-2097	MEDIUM	1.1.1n-0+deb11u3		openssl: AES
↳ OCB fails to encrypt some bytes					
↳ avd.aquasec.com/nvd/cve-2022-2097					



	CVE-2007-6755	LOW			Dual_EC_DRBG:
↳	weak pseudo random number generator				
					https://
↳	avd.aquasec.com/nvd/cve-2007-6755				

	CVE-2010-0928				openssl: RSA
↳	authentication weakness				
					https://
↳	avd.aquasec.com/nvd/cve-2010-0928				

	openssl	CVE-2022-2097	MEDIUM		openssl: AES
↳	OCB fails to encrypt some bytes				
					https://
↳	avd.aquasec.com/nvd/cve-2022-2097				

	CVE-2007-6755	LOW			Dual_EC_DRBG:
↳	weak pseudo random number generator				
					https://
↳	avd.aquasec.com/nvd/cve-2007-6755				

	CVE-2010-0928				openssl: RSA
↳	authentication weakness				
					https://
↳	avd.aquasec.com/nvd/cve-2010-0928				

↳					

