

Security Auditing Report

Teleport Cloud

Prepared for: Gravitational, Inc. DBA Teleport
Prepared by: Ben Caller, Norbert Szetei
Date: 10/10/2022

© DOYENSEC

Table of Contents

Table of Contents	1
Revision History	2
Contacts	2
Executive Summary	3
Methodology	4
Project Findings	6
Appendix A - Vulnerability Classification	37
Appendix B - Remediation Checklist	38
Appendix C - Retesting Results	39

Revision History

Version	Date	Description	Author
1	3/18/2022	First release of the final report	Norbert Szetei, Ben Caller
2	3/20/2022	Peer Review	Norbert Szetei
3	3/22/2022	Peer Review	Lorenzo Stella
4	3/26/2022	Peer Review	Luca Carettoni
5	10/10/2022	Retesting results	Lorenzo Stella

Contacts

Company	Name	Email
Gravitational, Inc	Reed Loden	reed@goteleport.com
Gravitational, Inc	Russell Jones	rjones@gravitational.com
Doyensec, LLC	Luca Carettoni	luca@doyensec.com
Doyensec, LLC	John Villamil	john@doyensec.com

Executive Summary

Overview

Gravitational, Inc. (DBA "Teleport") engaged Doyensec to perform a security assessment of the Teleport Cloud platform. The project commenced on 2/28/2022 and ended on 3/18/2022 requiring 2 security researcher. The project resulted in eleven (11) findings, with no critical or high severity findings.

The project consisted of a manual web application security assessment.

Testing was conducted remotely from Doyensec's EMEA and US offices.

On 10/03/2022, Doyensec retested the platform. This deliverable represents the state of all discovered vulnerabilities as of 10/10/2022.

Scope

Through meetings with Teleport the scope of the project was clearly defined. The agreed upon assets are listed below:

- Teleport Cloud

The testing took place in a staging environment using the latest version of the software at the time of testing. In detail, this activity was performed on the following releases:

- gravitational/cloud
  - 42b893e5434e41b24b43b2cf5c6fa5f6ce4cd966
- gravitational/teleport
  - 6e3b328a7ec5bea47c39ca6655f61e950dac8ed9
- gravitational/okta-terraform
  - 8248c7c5eeec4492845141e8f5ff4e6c884310ef

Scoping Restrictions

During the engagement, Doyensec did not encounter difficulties with testing.

Findings Summary

Doyensec researchers discovered and reported eleven (11) vulnerabilities in the Teleport Cloud platform. While most of the issues are departure from best practices and low-severity flaws, Doyensec identified three issues rated as medium.

It is important to reiterate that this report represents a snapshot of the environment's security posture at a point in time.

The findings included denial of service vulnerabilities, a Teleport audit bypass, and cloud misconfigurations.

Overall, the security posture of the Internet-facing APIs was found to be in line with industry best practices.

At the design level, Doyensec found the system to be well architected.

Recommendations

The following recommendations are proposed based on studying Teleport Cloud's security posture and the vulnerabilities discovered during this engagement.

Short-term improvements

- Work on mitigating the discovered vulnerabilities. You can use **Appendix B** - Remediation Checklist to make sure that you have covered all areas.

Methodology

Overview

Doyensec treats each engagement as a fluid entity. We use a standard base of tools and techniques from which we built our own unique methodology. Our 30 years of information security experience has taught us that mixing offensive and defensive philosophies is the key to standing against threats. Thus we recommend a *whitebox* approach combining dynamic fault injection with an in-depth study of the source code to maximize the ROI on bug hunting.

During this assessment, we have employed standard testing methodologies (e.g., OWASP Testing guide recommendations), as well as custom checklists, to ensure full coverage of both code and vulnerability classes.

Setup Phase

Gravitational, Inc. provided access to the online staging environment (platform.teleport.sh with access to Kubernetes and Sales Center), staging AWS accounts and source code repositories.

Tooling

When performing assessments, we combine manual security testing with state-of-the-art tools in order to improve efficiency and efficacy of our effort.

During this engagement, we used the following tools:

- Burp Suite
- CloudMapper
- go-fuzz
- Semgrep
- Curl, netcat and other Linux utilities

Web Application and API Techniques

Web assessments are centered around the data sent between clients and servers. In this realm, the principle audit tool is Burp Suite. However, we also use a large set of custom scripts and extensions to perform specific audit tasks. We focus on authorization, authentication, integrity and trust. We study how data is interpreted, parsed, stored, and relayed between producers and consumers.

We subvert the client with malicious data through reflected and DOM based Cross Site Scripting and by breaking assumptions in trust. We test the server endpoints for injection style flaws including, but not limited to, SQL, template, XML, and command injection flaws. We look at each request and response pair for potential Cross Site Request Forgery and race conditions. We study the application for subtle logic issues, whether they are authorization bypasses or insecure object references. Session storage and retrieval is scrutinized and user separation is thoroughly tested.

Web security is not limited to popular bug titles. Doyensec researchers understand the goals and needs of the application to find ways of breaking the assumed control flow.

Cloud Configuration Review

Cloud configuration reviews are centered around the evaluation of security measures that the client implements and are part of their responsibilities in the shared security model. In this realm, we identify assets, define boundaries and carefully review all applicable settings to uncover misconfigurations and departures from best practices.

By combining tooling (Nessus, ScoutSuite, G-Scout, Forseti Security and custom scripts) with manual review skills, Doyensec can perform in-

depth reviews that will expose concrete risks and suggest actionable mitigations.

Our reviews can focus on the following areas:

- Governance
- Network Configuration and Management
- Asset Configuration and Management
- Logical Access Control
- Data Encryption
- Security Logging and Monitoring
- Security Incident Response
- Disaster Recovery

Examples of identifiable misconfigurations, weaknesses and departures from best practices include, but are not limited to:

- Sensitive services or databases exposed to the Internet
- Use of plain-text protocols
- Lack of network ACLs (Inbound)
- Lack of network ACLs (Outbound)
- Poor key management and rotation practices
- Lack of Multi-Factor Authentication
- Weak user policies
- Poorly maintained passwords and access keys
- World-readable buckets
- World-writable buckets
- Insecure network design and operational practices that could facilitate data exfiltration or lateral movement
- Missing network isolation or segregation layers

Project Findings

The table below lists the findings with their associated ID and severity. The severity ranking and vulnerability classes are defined in **Appendix A** at the end of this document. The vulnerability class column groups the entry into a common category, while the status column refers to whether the finding has been fixed at the time of writing.

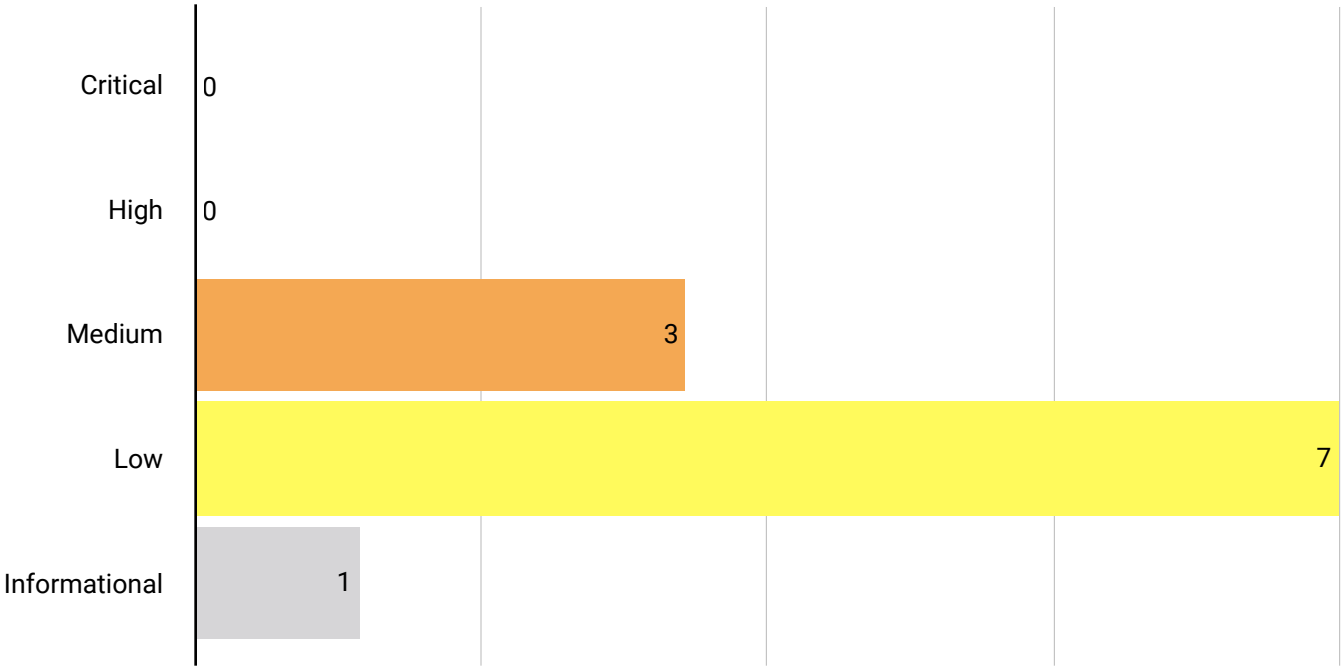
This table is organized by time of discovery. The issues at the top were found first, while those at the bottom were found last. Presenting the table in this fashion has a number of benefits. It inherently shows the path our auditing took through the target and may also reveal how easy or difficult it was to discover certain findings. As a security engagement progresses, the researchers will gain a deeper understanding of a target which is also shown in this table.

Findings Recap Table

ID	Title	Vulnerability Class	Severity	Status
TELC-Q122-1	Authenticated Denial of Service in Sales Center	Denial of Service (DoS)	Low	Closed
TELC-Q122-2	CSV Injection Via S3 Data Export Using Sales Center Service	Injection Flaws (SQL, XML, Command, Path, etc)	Informational	Risk Accepted
TELC-Q122-3	S3 Buckets Allow Insecure Transport	Security Misconfiguration	Low	Closed
TELC-Q122-4	Private S3 Buckets Without Public Access Block	Security Misconfiguration	Low	Closed
TELC-Q122-5	S3 Buckets With Encryption Disabled	Security Misconfiguration	Low	Closed
TELC-Q122-6	Tenant IAM Policy Too Broad	Insecure Design	Medium	Closed
TELC-Q122-7	Wildcard Resource Permissions in Staging Role	Security Misconfiguration	Medium	Closed
TELC-Q122-8	Denial Of Service Via Malicious MySQL COM_CHANGE_USER Packet	Denial of Service (DoS)	Low	Closed
TELC-Q122-9	Teleport Allows Connection To Arbitrary SSH server	Insecure Design	Low	Risk Accepted
TELC-Q122-10	Denial Of Service Via WebSocket Upgrader	Denial of Service (DoS)	Medium	Closed
TELC-Q122-11	Database Query Audit Bypass	Insecure Design	Low	Closed

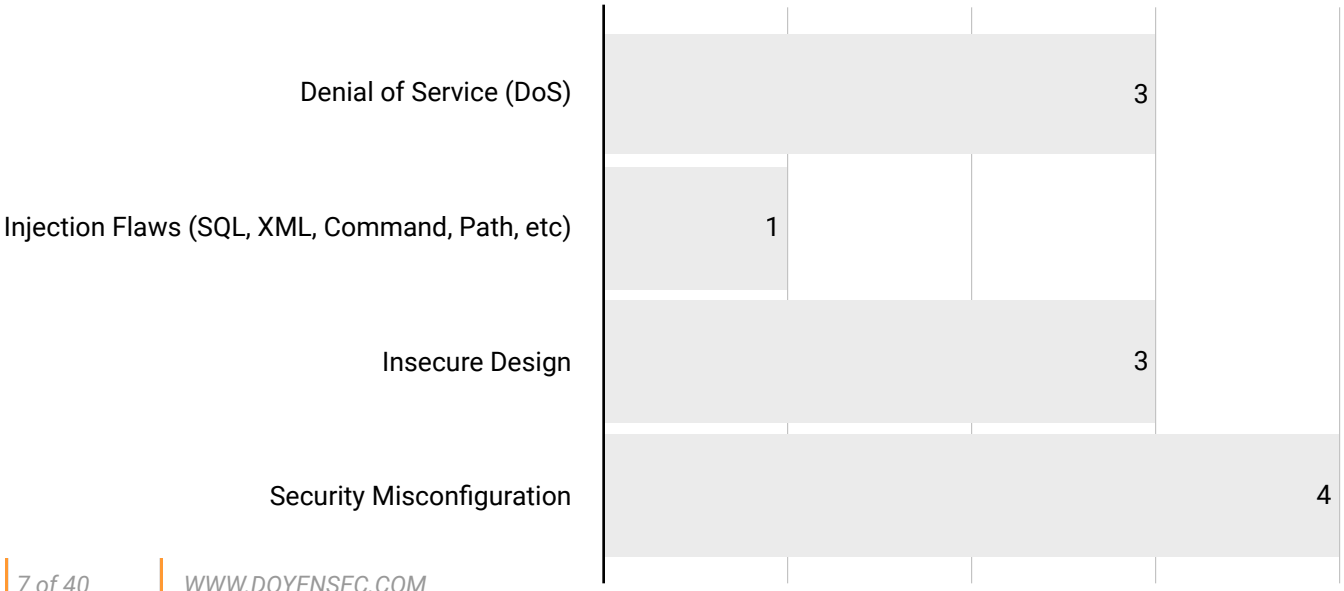
Findings per Severity

The table below provides a summary of the findings per severity.



Findings per Type

The table below provides a summary of the findings per vulnerability class.



TELC-Q122-1. Authenticated Denial of Service in Sales Center	
Severity	**Low**
Vulnerability Class	Denial of Service (DoS)
Component	api/sales/v1/sales.pb.go
Status	Closed

Description

Sales Center panics when it receives invalid protobuf messages. The `sales.pb.go` file, which is auto-generated by `protoc-gen-gogo` (https://github.com/gogo/protobuf), does not perform proper bounds checks or handle errors gracefully.

Sending 5 null bytes (an empty `grpc-web+proto` request) to any of the following RPC endpoints will kill Sales Center with a segfault.

- GetPlan
- GetSubscription
- CreatePlan
- CreateProduct
- CreateReferral
- UpdateAccount
- UpdateProduct
- UpdateSubscription

This can be used for Denial of Service (DoS), as Sales Center will be unavailable while it restarts.

Reproduction Steps

Make an authenticated RPC request with 5 null bytes in the body.

```
curl -is -XPOST \
  -H 'Content-Type: application/grpc-web+proto'
  -b '__Host-grv_app_session=6bb3219...' \
  --data-binary $'\x00\x00\x00\x00\x00' \
  https://sales-center-staging.platform.teleport.sh/
gravitational.cloud.sales.v1.SalesService/GetPlan
```

Sales Center panics and segfaults trying to access a near-zero memory address (0x60).

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x60 pc=0x1b19766]

goroutine 1147 [running]:
```

```
github.com/gravitational/cloud/api/sales/v1.
(*Plan).MarshalToSizedBuffer(0x0, ...)
    /gopath/src/github.com/gravitational/cloud/api/sales/v1/
sales.pb.go:7091 +0x26
github.com/gravitational/cloud/api/sales/v1.(*Plan).Marshal(0x0, ...)
    /gopath/src/github.com/gravitational/cloud/api/sales/v1/
sales.pb.go:7074 +0x7a
google.golang.org/protobuf/internal/impl.legacyMarshal(0x236d9c8, ...)
    /gopath/pkg/mod/google.golang.org/protobuf@v1.27.1/internal/impl/
legacy_message.go:404 +0xb3
google.golang.org/protobuf/proto.MarshalOptions.marshal(0xc000000001, ...)
    /gopath/pkg/mod/google.golang.org/protobuf@v1.27.1/proto/encode.go:163 +0x25e
google.golang.org/protobuf/proto.MarshalOptions.MarshalAppend(0x1000001, ...)
...
```

Similarly with `CreatePlan`:

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x0 pc=0x1b51f1d]

goroutine 859 [running]:
github.com/gravitational/cloud/pkg/salesserver/sales.
(*handler).CreatePlan(0xc0004b4300, ...)
 /gopath/src/github.com/gravitational/cloud/pkg/salesserver/sales/handler.go:130
+0x3d
github.com/gravitational/cloud/api/sales/
v1._SalesService_CreatePlan_Handler.func1(0x2350480, ...)
 /gopath/src/github.com/gravitational/cloud/api/sales/v1/sales.pb.go:5674 +0x89
github.com/gravitational/cloud/pkg/salesserver.
(*Server).withAuthToken.func1(0x2350480, ...)
 /gopath/src/github.com/gravitational/cloud/pkg/salesserver/middleware.go:43
+0x3d7
google.golang.org/grpc.chainUnaryInterceptors.func1.1(0x2350480, ...)
 /gopath/pkg/mod/google.golang.org/grpc@v1.41.0/server.go:1113 +0x8c
```

And with `CreateProduct`:

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x10 pc=0x1b52a8a]

goroutine 1286 [running]:
github.com/gravitational/cloud/pkg/salesserver/sales.asProductParams(...)
 /gopath/src/github.com/gravitational/cloud/pkg/salesserver/sales/
converters.go:81
github.com/gravitational/cloud/pkg/salesserver/sales.
(*handler).CreateProduct(0xc0008164f0, ...)
 /gopath/src/github.com/gravitational/cloud/pkg/salesserver/sales/handler.go:179
+0x6a
github.com/gravitational/cloud/api/sales/
v1._SalesService_CreateProduct_Handler.func1(0x2350480, ...)
```

Impact

Medium. An attacker can kill Sales Center by causing a panic. Sales Center will be unavailable for a short period of time while it is restarted.

Complexity

Medium. Basic web security skills are required. The main barrier is that the attacker must be authenticated to Sales Center.

Remediation

Apply sanity-checks and error handling to RPC calls.

As mentioned before, the `sales.pb.go` file is auto-generated by `protoc-gen-gogo`. This library may be inappropriate for exposed services which may receive malformed or malicious requests. Gogo has configurable "speeds" and it is possible that different settings generate code without the vulnerability. If not, a different protobuf / gRPC library or JSON requests should be used instead.

Resources

- "Nil Pointer in MarshalTo/MarshalToSizedBuffer" - unresolved Github issue
  https://github.com/gogo/protobuf/issues/651

TELC-Q122-2. CSV Injection Via S3 Data Export Using Sales Center Service	
Severity	**Informational**
Vulnerability Class	Injection Flaws (SQL, XML, Command, Path, etc)
Component	Sales Center
Status	Risk Accepted

Description

CSV Injection or Formula Injection[1] vulnerabilities occur when embedding untrusted user input inside `csv` files. Any cells starting with equal (=), plus (+), minus (-), or "at" (@) characters will be interpreted as a formula by the spreadsheet software (like Microsoft Excel or Libre Office), turning a field into active content.

Sales Center exports the list of customers without sanitizing the data. Since the finding requires the permissions to create a new account or modify an existing one, we are lowering the severity as informational.

Reproduction Steps

The following request can be used to send the CSV injection data to the platform.

1. Visit https://sales-center-staging.platform.teleport.sh/accounts
2. Click on the *Add Account* button or edit an existing entry.
3. Insert a mathematical formula for the *Company Name*, *Contact Name*, *Street Addresses*, *City*, *State*, or *Postal Code* fields. In our case, we created an entry with the Company Name `=7*191`.
4. Save the entry and visit https://sales-center-staging.platform.teleport.sh/settings to verify the *S3 Export URI*.
5. Visit https://s3.console.aws.amazon.com/s3/buckets/int001-salesforce?prefix=input/&region=us-west-2 and download the CSV file. Upload happens in short periodic intervals, and in our case, the destination file was stored as `cloud-usage-report.csv`.
6. Observe that there are no changes in the user-submitted data, and consequently, they are interpreted. In the picture below, you can see a screenshot from Apple Numbers.

1125e9e4-a8c7-4b11-9cea-9aa5c3692ea3	dd-test2	dd-test2@doyensec.com
bbf41895-412d-4c1c-b74e-11916ac94c8f	1337	testing8@doyensec.com
4f52c61a-6c81-4b2f-84c8-88a1941e85c2	dd-test7-company	testing7@doyensec.com

1 https://www.owasp.org/index.php/CSV_Excel_Macro_Injection

Impact

Potentially high. If an attacker can coerce a user into downloading, opening, and executing the active content, then formula injection can result in the following potential dangers.

1. Leveraging an exploit against the actual spreadsheet software can result in the compromise of the machine opening the affected file
2. Taking advantage of the trust relationship between the end-user and the spreadsheet software in order to make the user click and enable Macros and Formulas. This will result in the compromise of the machine opening the affected file
3. Exfiltrating content from the spreadsheet

Complexity

High. Even if injecting malicious payloads is trivial, the attacker would need to coerce the user to download, open, and finally execute the active content in order to successfully exploit the vulnerability. Inserting the payload requires the permissions to create or modify accounts.

Recent versions of Microsoft Excel warn the user before executing active content.

Remediation

Fields that begin with Equals (=), Plus (+), Minus (-), or At (“@“) characters should be prepended by a single quote (‘) or apostrophe (') character. Microsoft Excel will preserve data integrity by hiding this character when rendering the spreadsheet and will not trigger the active content.

Resources

- "CSV Injection", OWASP Community
  https://owasp.org/www-community/attacks/CSV_Injection

- "Export to RCE", Adam Greenhill
  https://owasp.org/www-chapter-toronto/assets/slides/ExporttoRCE.pdf

TELC-Q122-3. S3 Buckets Allow Insecure Transport	
Severity	**Low**
Vulnerability Class	Security Misconfiguration
Component	AWS S3
Status	Closed

Description

By default S3 buckets and objects can be accessed both over HTTPS (TLS) connections and unencrypted HTTP connections. An attacker suitably positioned to view the network traffic of a legitimate user or Teleport Cloud service could eavesdrop on or manipulate their unencrypted traffic. The attacker would be able to obtain sensitive information, alter data stored in S3 and alter data returned to the user.

In staging, all buckets including `teleport-cloud-staging-us-west-2-audit-sessions` are affected.

Buckets are affected if they have no bucket policy, or if the policy does not deny access when `aws:SecureTransport` is false.

Reproduction Steps

Download an object from `teleport-cloud-staging-us-west-2-audit-sessions` over HTTP (non-TLS) using the following example link:

http://teleport-cloud-staging-us-west-2-audit-sessions.s3.amazonaws.com/tenant-vunib-situl-higuk-hifap-x2o7zxvec6vbi/hello.txt

The bucket policy can be retrieved with:

```
aws s3api get-bucket-policy --bucket teleport-cloud-staging-us-west-2-audit-sessions
```

Impact

High. This flaw could expose sensitive data and allow attackers to alter data uploaded to S3.

Complexity

This issue highlights a missing security hardening configuration, rather than a vulnerability per se. Teleport and most software will use the TLS transport by default, so accidental use of unencrypted communication is unlikely.

Remediation

Set a bucket policy which denies access if `aws:SecureTransport` **is false**, e.g.:

```
{
  "Version": "2008-10-17",
  "Statement": [
      {
          "Effect": "Deny",
          "Principal": "*",
          "Action": "s3:*",
          "Resource": [
              "arn:aws:s3:::teleport-cloud-staging-us-west-2-audit-sessions",
              "arn:aws:s3:::teleport-cloud-staging-us-west-2-audit-sessions/*"
          ],
          "Condition": {
              "Bool": {
                  "aws:SecureTransport": "false"
              }
          }
      }
  ]
}
```

Resources

- Amazon S3 Security Best Practices
  https://docs.aws.amazon.com/AmazonS3/latest/userguide/security-best-practices.html#transit

- "M3: Insecure Communication", OWASP Mobile Top 10
  https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication

- What S3 bucket policy should I use to comply with the AWS Config rule s3-bucket-ssl-requests-only?
  https://aws.amazon.com/premiumsupport/knowledge-center/s3-bucket-policy-for-config-rule/

TELC-Q122-4. Private S3 Buckets Without Public Access Block	
Severity	**Low**
Vulnerability Class	Security Misconfiguration
Component	AWS S3
Status	Closed

Description

Each object in an S3 bucket has its own ACL. If a service attaches public ACLs to objects, the objects will be publicly readable even if the bucket itself is private. While there are legitimate use-cases for this, it is often caused accidentally. For buckets where no files should be public, this can be prevented with a public access block. The block can be configured to ignore public ACLs and to cause requests to fail if they would make objects public.

Ensure that every S3 bucket which is not designed to contain public objects has a public access block.

This affects the following buckets in staging:

- `cf-templates-1xpi0n38tjizw-us-west-2`
- `gravity-images20210107043806017200000001`
- `int001-salesforce`
- `releases-service20210701152909403200000001`
- `teleport-cloud-staging-us-west-2-audit-sessions`

In the AWS console these can be identified as the Access column contains "*Objects can be public*".

Reproduction Steps

Put an object with a publicly readable ACL into a private bucket e.g.

```
aws s3api put-object \
  --bucket BUCKET_NAME --key public.txt --body public.txt --acl public-read
```

Then retrieve the file via `https://BUCKET_NAME.s3.amazonaws.com/public.txt`. Note that no signature or access key was required. An example is:

http://teleport-cloud-staging-us-west-2-audit-sessions.s3.amazonaws.com/tenant-vunib-situl-higuk-hifap-x2o7zxvec6vbi/hello.txt

Finding out which files in a bucket are public requires accessing each object in turn.

Impact

High. If a public ACL gets attached to an object, sensitive data could be exposed.

Complexity

This issue highlights a missing security hardening configuration, rather than a vulnerability per se. There are attackers constantly scanning S3 buckets trying to retrieve public files.

Remediation

Add a public access block to all buckets which should not contain any public objects.

Example terraform configuration:

```
resource "aws_s3_bucket_public_access_block" "s3_public_access_block" {
  bucket = aws_s3_bucket.my_bucket.id

  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}
```

Resources

- Blocking public access to your Amazon S3 storage
  https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-control-block-public-access.html

- There's a hole in my bucket - National Cyber Security Centre
  https://www.ncsc.gov.uk/blog-post/theres-hole-my-bucket

- Configuring block public access settings for your S3 buckets
  https://docs.aws.amazon.com/AmazonS3/latest/userguide/configuring-block-public-access-bucket.html

TELC-Q122-5. S3 Buckets With Encryption Disabled	
Severity	**Low**
Vulnerability Class	Security Misconfiguration
Component	AWS S3
Status	Closed

Description

Server-Side Encryption (SSE) provides fully-managed encryption of all objects uploaded to an S3 bucket. It encrypts objects before storing them on disks in AWS data centers, and decrypts them when downloading the objects.

The following 3 S3 buckets are not set up with Server-Side Encryption:

- `int001-salesforce`
- `releases-service20210701152909403200000001`
- `teleport-cloud-staging-us-west-2-audit-sessions`
- `teleport-cloud-terraform-state-09-21`
- `teleport-releases-staging`

Reproduction Steps

S3 bucket policies and encryption can be verified using the AWS CLI command:

```
aws s3api get-bucket-encryption \
  --bucket teleport-cloud-staging-us-west-2-audit-sessions

aws s3api get-bucket-policy --bucket teleport-cloud-staging-us-west-2-audit-sessions
```

Impact

Buckets and objects encryption ensures confidentiality and integrity against advanced attackers, rogue Amazon sysadmin operators and the risk from decommissioned and discarded AWS hardware.

Complexity

These missing settings are considered departures from best practices, and cannot be abused without a foothold on the AWS infrastructure or access to decommissioned AWS hardware.

Remediation

Enable default Server-Side Encryption on all S3 buckets.

Resources

- "Amazon S3 default encryption for S3 buckets", AWS Docs
  https://docs.aws.amazon.com/AmazonS3/latest/dev/bucket-encryption.html

- "How to Prevent Uploads of Unencrypted Objects to Amazon S3", AWS Docs
  https://aws.amazon.com/blogs/security/how-to-prevent-uploads-of-unencrypted-objects-to-amazon-s3/

TELC-Q122-6. Tenant IAM Policy Too Broad	
Severity	**Medium**
Vulnerability Class	Insecure Design
Component	AWS IAM policy tenant-*
Status	Closed

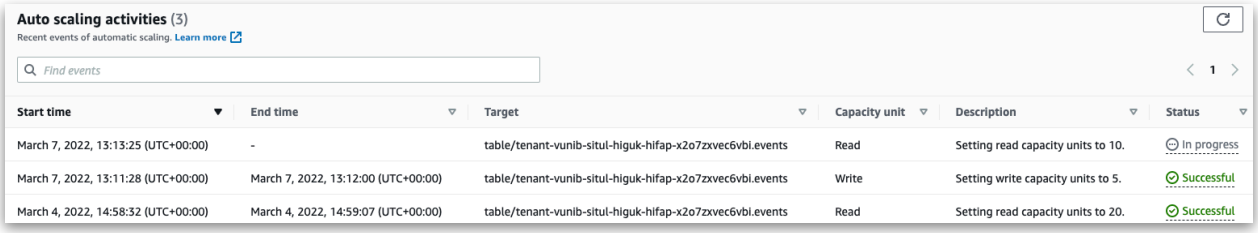
Description

As described in TEL-Q121-7, an SSRF vulnerability exists which allows tenants to retrieve AWS access keys from `kiam` for their tenant's role. When tenants have access to AWS, it is especially important to lock down the IAM permissions to prevent cross-tenant access and abuse of costly API calls.

The policy statement for auto-scaling allows cross-tenant modifications.

```
{
  "Action": [
    "application-autoscaling:PutScalingPolicy",
    "application-autoscaling:RegisterScalableTarget"
  ],
  "Effect": "Allow",
  "Resource": "*"
}
```

Since the Application Auto Scaling APIs do not support specifying resource ARNs, it is not possible to limit these actions to a tenants own resources. This policy allows any tenant to affect the auto-scaling policy of any other tenant's DynamoDB tables.



If the account were to contain a service-linked IAM role for auto-scaling of other services such as RDS, EMR, ECS or Lambda, this tenant policy would allow alteration of the autoscaling for those resources. In the staging account, however, only DynamoDB has an AWS Service Role for Application Auto-scaling, so only DynamoDB can be affected.

While preventing cross-tenant access, the permissions for S3 and DynamoDB are still too broad and allow abuse of the cloud infrastructure.

```
{
```

```
      "Action": [
        "s3:*Object*",
        "s3:*Multipart*",
        "dynamodb:*"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:dynamodb:us-west-2:599519581022:table/tenant-xyz.*",
        "arn:aws:s3:::teleport-cloud-staging-us-west-2-audit-sessions/tenant-xyz",
        "arn:aws:s3:::teleport-cloud-staging-us-west-2-audit-sessions/tenant-xyz/*"
      ]
    }
```

On the S3 side, this policy allows:

- setting arbitrary ACLs on objects, such as granting anonymous users full control
- storing and retrieving files from expensive storage classes such as Glacier

In DynamoDB, this policy allows:

- deletion of the tenant's own tables
- creation of unlimited tables as long as the bucket name has the tenant prefix
- backup creation
- exporting to the S3 bucket (under the tenant prefix)
- setting the table's tags

Reproduction Steps

The first step is to retrieve temporary tenant credentials either using the SSRF as described in TEL-Q121-7 or via shell access to the auth server and the following bash command (as `curl` is not available):

```
exec 3<>/dev/tcp/169.254.169.254/80 && echo -e "GET /latest/meta-data/iam/
security-credentials/tenant-xyz HTTP/1.1\r\nhost: 169.254.169.254\r\n\r\n" >&3 &&
cat <&3
```

To put a publicly readable file in S3 in the One Zone IA storage class with a publicly writeable ACL, encryption and a tag:

```
aws s3api put-object \
  --grant-full-control uri=http://acs.amazonaws.com/groups/global/AllUsers \
  --bucket teleport-cloud-staging-us-west-2-audit-sessions \
  --key tenant-xyz/ONEZONE_IA.txt \
  --storage-class ONEZONE_IA \
  --body hello.txt \
  --server-side-encryption AES256 \
  --tagging Doyensec=d0Y3ns3c

{
    "ETag": "\"b1946ac92492d2347c6235b4d2611184\"",
    "ServerSideEncryption": "AES256"
}
```

e.g. http://teleport-cloud-staging-us-west-2-audit-sessions.s3.us-west-2.amazonaws.com/tenant-fifof-motaj-ranop-hifof-vxb65rgct6sgu/ONEZONE_IA.txt?acl

To alter the auto-scaling configuration of another tenant:

```
aws --region us-west-2 application-autoscaling register-scalable-target \
  --service-namespace dynamodb \
  --resource-id table/tenant-ben.whatever \
  --scalable-dimension "dynamodb:table:ReadCapacityUnits" \
  --min-capacity 1 \
  --max-capacity 10
```

Or (creating associated CloudWatch alarms):

```
aws --region us-west-2 application-autoscaling put-scaling-policy \
  --policy-name my-asp \
  --policy-type TargetTrackingScaling \
  --target-tracking-scaling-policy-configuration file://scale.json \
  --service-namespace dynamodb \
  --resource-id table/tenant-ben.whatever \
  --scalable-dimension "dynamodb:table:WriteCapacityUnits"

{
    "PolicyARN": "arn:aws:autoscaling:us-
west-2:599519581022:scalingPolicy:be47c90d-28e5-4ca0-a12e-315b76095063:resource/
dynamodb/table/tenant-ben.whatever:policyName/my-asp",
    "Alarms": [
        {
            "AlarmName": "TargetTracking-table/tenant-ben.whatever-
AlarmHigh-2cdfe529-bfa6-40c2-8a22-3b4789ade70c",
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:599519581022:alarm:TargetTracking-table/tenant-ben.whatever-
AlarmHigh-2cdfe529-bfa6-40c2-8a22-3b4789ade70c"
        },
        ...
```

where `scale.json` contains a simple target tracking scaling policy configuration such as:

```
{
  "TargetValue": 10.0,
  "PredefinedMetricSpecification": {
    "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
  }
}
```

To modify the DynamoDB table tags (not cross-tenant):

```
aws --region us-west-2 dynamodb tag-resource \
  --resource-arn arn:aws:dynamodb:us-west-2:599519581022:table/tenant-fifof-
motaj-ranop-hifof-vxb65rgct6sgu.main \
  --tags 'Key=cloud.gravitational.io/tenant,Value=:/=@+'
```

Impact

Due to the SSRF, tenants can retrieve AWS keys and make their own AWS requests. The IAM policy allows tenants to create unlimited DynamoDB tables with their tenant prefix and store files in different S3 storage classes. This may lead to excessive cloud bills. The auto-scaling configuration can be changed cross-tenant to affect any DynamoDB table in the account. This can be used for Denial of Service (DoS) against other tenants by decreasing their provisioned read/write capacity. By increasing provisioned capacity it could inflate Teleport's cloud bills.

Complexity

Medium. As described in TEL-Q121-7, retrieving AWS keys requires the attacker to be administrator of a tenant and set up a malicious OIDC provider. While tenants can find out their own UID easily (via the IAM role name or S3 location of session recordings), we do not currently have a way to leak the tenant IDs of other tenants. As it appears in the DynamoDB table name, the auto-scaling API calls cannot be made without knowing the victim tenant's tenant ID.

Remediation

All provisioning, deletion, scaling, tagging and upgrading tasks should be performed exclusively by the tenant operator rather than the tenant itself.

Remove the application-autoscaling access. In the cloud, auto-scaling policy should be controlled by the tenant operator.

Limit the allowed DynamoDB actions. Tenants should only be able to fetch and store data, not create tables or alter table configuration. The policy could, for instance, be limited to:

- `dynamodb:*Item`
- `dynamodb:Query`
- `dynamodb:Scan`

Restrict the allowed S3 storage classes with a Condition on the bucket or IAM policy:

```
"StringEquals": {
  "s3:x-amz-storage-class": [
    "STANDARD"
  ]
}
```

Preventing S3 objects being made public is best done with a public access block as described in TELC-Q122-4, but can also be done via a bucket or IAM policy.

While not harmful, the `iam:CreateServiceLinkedRole` part of the policy can also be removed as long as the account has been provisioned with the DynamoDB Auto-scaling role.

Fix the SSRF vulnerability or otherwise prevent customers from retrieving AWS keys.

Resources

- Actions, resources, and condition keys for AWS Application Auto Scaling
  https://docs.aws.amazon.com/service-authorization/latest/reference/list_awsapplicationautoscaling.html
- Using Amazon S3 storage classes
  https://docs.aws.amazon.com/AmazonS3/latest/userguide/storage-class-intro.html
- Amazon S3 condition key examples
  https://docs.aws.amazon.com/AmazonS3/latest/userguide/amazon-s3-policy-keys.html
- TEL-Q121-7. Server-Side Request Forgery via OpenID Connect
  https://www.doyensec.com/resources/teleport-cloud-audit-q1-2021.pdf

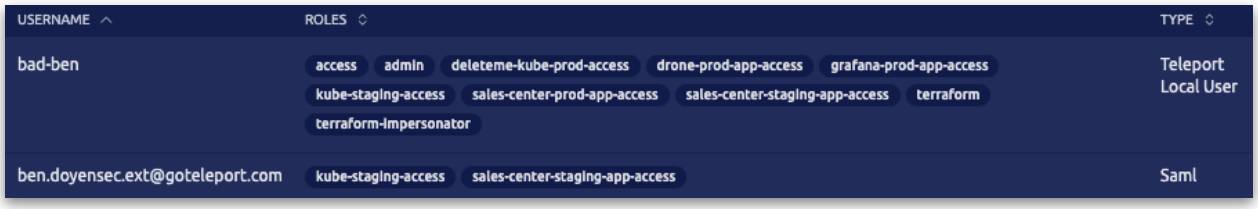
TELC-Q122-7. Wildcard Resource Permissions in Staging Role	
Severity	**Medium**
Vulnerability Class	Security Misconfiguration
Component	kube-staging-access deploy/terraform/modules/platform-roles/kube-staging-access.tf
Status	Closed

Description

The `kube-staging-access` role is designed to give access to the Teleport Cloud staging environment. The role definition is defined in deploy/terraform/modules/platform-roles/kube-staging-access.tf:

```
allow {
  kubernetes_groups = [
    "system:masters"
  ]
  rules {
    resources = ["*"]
    verbs     = ["*"]
  }
  kubernetes_labels {
    key   = "cluster"
    value = ["teleport-cloud-staging-us-west-2"]
  }
}
```

The wildcard on resources and verbs allows roles and users to be edited, effectively granting admin privileges. Doyensec was able to create a new user with several additional roles:



This was fixed in https://github.com/gravitational/cloud/pull/1439/files.

Reproduction Steps

The role allowed creation and modification of users and roles from the web interface.

Impact

High. Users could escalate privileges to administrator level. This would allow creation and modification of users, roles and auth connectors, and access to all resources protected by Teleport such as Grafana prod.

Complexity

High. Escalating privilege is trivial, but the attacker must first be granted the `kube-staging-access` role in the platform.teleport.sh cluster.

Remediation

This was fixed in https://github.com/gravitational/cloud/pull/1439/files by removing the `rules` section.

TELC-Q122-8. Denial Of Service Via Malicious MySQL COM_CHANGE_USER Packet	
Severity	**Low**
Vulnerability Class	Denial of Service (DoS)
Component	lib/srv/db/mysql/protocol/command.go#L102
Status	Closed

Description

Teleport Database Access supports multiple database engines, including MySQL. For this, it uses an external package https://github.com/go-mysql-org/go-mysql. However, during the protocol packet reading, Teleport additionally implements several parsing routines.

We reviewed them thoroughly and identified a logical off-by-one error when parsing the `COM_CHANGE_USER` packet (type `0x11`). By sending a specially crafted packet, it is possible to cause a runtime error, resulting in panic.

Reproduction Steps

As you can see in lib/srv/db/mysql/protocol/command.go#88 below, the `idx` variable contains the return value of the `bytes.IndexByte` function. According to the Golang documentation, if the null byte is not present in the payload, the `idx` is set to `-1`. That means that the condition `idx < -1` is **never** evaluated as `true`.

Consequently, if the null byte is not found, the array access for `packetBytes` causes the starting index to be bigger than the end index, resulting in panic.

```
func parseChangeUserPacket(packetBytes []byte) (Packet, error) {
    if len(packetBytes) < packetHeaderAndTypeSize {
        return nil, trace.BadParameter("failed to parse COM_CHANGE_USER packet:
%v", packetBytes)
    }

    // User is the first null-terminated string in the payload:
    // https://dev.mysql.com/doc/internals/en/com-change-user.html#packet-
COM_CHANGE_USER
    idx := bytes.IndexByte(packetBytes[packetHeaderAndTypeSize:], 0x00)
    if idx < -1 {
        return nil, trace.BadParameter("failed to parse COM_CHANGE_USER packet:
%v", packetBytes)
    }

    return &ChangeUser{
        packet: packet{bytes: packetBytes},
        user:   string(packetBytes[packetHeaderAndTypeSize :
packetHeaderAndTypeSize+idx]),
    }, nil
}
```

Add the following test inside the `lib/srv/db/mysql/protocol/packet_test.go` to trigger the error:

```
input: bytes.NewBuffer([]byte{
    0x01, 0x00, 0x00, 0x00, // header
    0x11, // type
}),
```

By running it inside the `teleport/lib/srv/db/mysql` directory, we can see the failing test:

```
$ go test ./protocol
--- FAIL: TestParsePacket (0.00s)
    --- FAIL: TestParsePacket/unrecognized_type (0.00s)
panic: runtime error: slice bounds out of range [5:4] [recovered]
 panic: runtime error: slice bounds out of range [5:4]

goroutine 40 [running]:
testing.tRunner.func1.2({0x12a9240, 0xc000210000})
 /usr/local/opt/go/libexec/src/testing/testing.go:1209 +0x24e
testing.tRunner.func1()
 /usr/local/opt/go/libexec/src/testing/testing.go:1212 +0x218
panic({0x12a9240, 0xc000210000})
 /usr/local/opt/go/libexec/src/runtime/panic.go:1038 +0x215
github.com/gravitational/teleport/lib/srv/db/mysql/
protocol.parseChangeUserPacket({0xc00020e008, 0x5, 0x138a180})
 /Users/tbnz/.go/src/github.com/gravitational/teleport/lib/srv/db/mysql/
protocol/command.go:102 +0x217
github.com/gravitational/teleport/lib/srv/db/mysql/
protocol.ParsePacket({0x132c8e0, 0xc000115d10})
 /Users/tbnz/.go/src/github.com/gravitational/teleport/lib/srv/db/mysql/
protocol/packet.go:58 +0x62
github.com/gravitational/teleport/lib/srv/db/mysql/
protocol.TestParsePacket.func1(0x0)
 /Users/tbnz/.go/src/github.com/gravitational/teleport/lib/srv/db/mysql/
protocol/packet_test.go:334 +0x39
testing.tRunner(0xc0001b29c0, 0xc000128c00)
 /usr/local/opt/go/libexec/src/testing/testing.go:1259 +0x102
created by testing.(*T).Run
 /usr/local/opt/go/libexec/src/testing/testing.go:1306 +0x35a
FAIL    github.com/gravitational/teleport/lib/srv/db/mysql/protocol 0.010s
FAIL
```

Impact

Possibly high. An attacker can cause the denial of service of all components using the MySQL protocol and possibly terminate the teleport daemon.

Complexity

High. The attacker has to set up a malicious MySQL server and send the malformed packet back to the client.

Remediation

Fix the condition inside the `parseChangeUserPacket`**. It is sufficient to change**

```
idx := bytes.IndexByte(packetBytes[packetHeaderAndTypeSize:], 0x00)
if idx < -1 {
```

to

```
idx := bytes.IndexByte(packetBytes[packetHeaderAndTypeSize:], 0x00)
if idx < 0 {
```

TELC-Q122-9. Teleport Allows Connection To Arbitrary SSH server	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	Teleport Web Interface
Status	Risk Accepted

Description

Teleport administrators can add SSH nodes to the cluster, used by the allowed users for connecting to the trusted Teleport SSH or OpenSSH servers. Moreover, the roles could be restricted (e.g., editor) not to grant this access.

Doyensec discovered that it is possible to initialize the connection to an arbitrary server, effectively ignoring all mentioned restrictions.

Reproduction Steps

Log in to the web interface and visit the `https://SERVER/web/cluster/CLUSTER/console/node/HOST:PORT/USER`, where the capitalized names represent your cluster and additional values you have to substitute.

For example, using https://platform.teleport.sh/web/cluster/platform.teleport.sh/console/node/doyensec.com:1337/testing we can obtain the following connection from the AWS cluster, intercepted via `ncat`:

```
$ ncat -v -lp 1337
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::1337
Ncat: Listening on 0.0.0.0:1337
Ncat: Connection from 3.69.93.191.
Ncat: Connection from 3.69.93.191:17697.
```

Impact

The impact is low, and an authenticated attacker could automate the finding for port scanning of arbitrary hosts. There is a possibility that the IP running the teleport will be added to public blacklists due to unusual or suspicious traffic. The finding could be used for blind SSRF too.

Lastly, the finding could be used as a precursor for additional vulnerabilities, enlarging the attack surface. For example, the web-SSH session implemented via WebSockets was vulnerable to Denial of Service in the past, and we discovered that another bug was added in the forthcoming Teleport 9.x branch. See `TELC-Q122-6 Denial Of Service Via Multiple WebSocket Connections` or `TELC-Q122-10 Denial Of Service Via Socket Upgrader` for detailed information.

Complexity

Low. The attacker needs to have a Teleport web account with an arbitrary role.

Remediation

Disallow the SSH web client functionality without prior authorization.

Resources

- Teleport Documentation - Server Access Getting Started Guide
  https://goteleport.com/docs/server-access/getting-started/

TELC-Q122-10. Denial Of Service Via WebSocket Upgrader	
Severity	**Medium**
Vulnerability Class	Denial of Service (DoS)
Component	Teleport WebSocket Terminal Handler https://github.com/gravitational/teleport/blob/bac0ccdc99068702beba395e12de114e961058b6/lib/web/terminal.go#L198
Status	Closed

Description

When a Teleport user attempts to open a web-SSH session, the web browser sends the following HTTP request to initialize the WebSocket tunnel:

```
GET /v1/webapi/sites/lambda/connect?
access_token=5a093510d6838c58545c0e3eec60913dfb1a6be88d8812f9812ff3b393ae2d26&par
ams=%7B%22login%22:%22tbnz%22,%22sid%22:%22c15a2822-d74a-47d4-
bafd-4d30d126f8c1%22,%22server_id%22:%22<redacted>:1337%22,%22term%22:%7B%22h%22:
69,%22w%22:284%7D%7D HTTP/1.1
Host: lambda:3080
Connection: Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
Origin: https://lambda:3080
Sec-WebSocket-Version: 13
Cookie: <valid-session>
Sec-WebSocket-Key: <ws-key>
```

By removing the highlighted line and sending (or resending) the same request, a potentially malicious authenticated user can cause the WebSocket connection to fail. However, the error is only logged, and instead of rejecting the request, the application continues using the malformed WebSocket handler.

Consequently, the teleport daemon crashes with the following error:

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x10 pc=0x5cf9bd8]

goroutine 3222 [running]:
github.com/gorilla/websocket.(*Conn).WriteMessage(0x87446e0, 0xc000dd6000,
{0xc001cf9980, 0x166, 0x166})
    /Users/tbnz/.go/pkg/mod/github.com/gorilla/websocket@v1.4.2/conn.go:753
+0x38
github.com/gravitational/teleport/lib/web.(*TerminalHandler).write(0xc000d6bb80,
{0xc001cf9680, 0x166, 0xc00154de68}, 0x8735170)
    /Users/tbnz/.go/src/github.com/gravitational/teleport/lib/web/
terminal.go:579 +0x15f
```

```
github.com/gravitational/teleport/lib/web.
(*TerminalHandler).writeError(0xc0001d9340, {0x870b4c0, 0xc00227e420}, 0x15)
    /Users/tbnz/.go/src/github.com/gravitational/teleport/lib/web/
terminal.go:517 +0x1a5
github.com/gravitational/teleport/lib/web.
(*TerminalHandler).streamTerminal(0xc000d6bb80, 0xc000dc8900, 0xc00127e600)
    /Users/tbnz/.go/src/github.com/gravitational/teleport/lib/web/
terminal.go:416 +0x2f2
created by github.com/gravitational/teleport/lib/web.(*TerminalHandler).handler
    /Users/tbnz/.go/src/github.com/gravitational/teleport/lib/web/
terminal.go:249 +0x5a6
```

Reproduction Steps

The following code shows the bug where the error is caught and logged, but the further execution is unaffected:

```
func (t *TerminalHandler) Serve(w http.ResponseWriter, r *http.Request) {
    // This allows closing of the websocket if the user logs out before exiting
    // the session.
    t.ctx.AddClosers(t)
    defer t.ctx.RemoveCloser(t)

    upgrader := websocket.Upgrader{
        ReadBufferSize:  1024,
        WriteBufferSize: 1024,
        CheckOrigin:     func(r *http.Request) bool { return true },
    }

    ws, err := upgrader.Upgrade(w, r, nil)
    if err != nil {
        t.log.Errorf("Error upgrading to websocket: %v", err)
    }

    t.handler(ws, r)
}
```

Afterward, the terminal handler uses the WebSocket, which triggers the crash:

```
func (t *TerminalHandler) handler(ws *websocket.Conn, r *http.Request) {
    defer ws.Close()

    // Create a context for signaling when the terminal session is over.
    t.terminalContext, t.terminalCancel =
context.WithCancel(context.Background())

    // Create a Teleport client, if not able to, show the reason to the user in
    // the terminal.
    tc, err := t.makeClient(ws, r)
```

Impact

High. Teleport daemon crashes, requiring a restart. Note that the current stable version `v8.3.4` is not affected, but the bug is present in the 9.x branches, for example, in the `v9.0.1` release.

Complexity

Low. The attacker needs to have a Teleport web account, and only basic web application security skills are required.

Remediation

If the WebSocket Upgrade fails, immediately return an error.

For inspiration, the Remote Desktop handler inside `lib/web/desktop.go` does exactly that:

```
ws, err := upgrader.Upgrade(w, r, nil)
if err != nil {
 return trace.Wrap(err)
}
```

Resources

- Commit Implement Moderated Sessions (#8563)
  https://github.com/gravitational/teleport/commit/ea810d30d99f26e58a190edc5facfbe0c09ea5e5

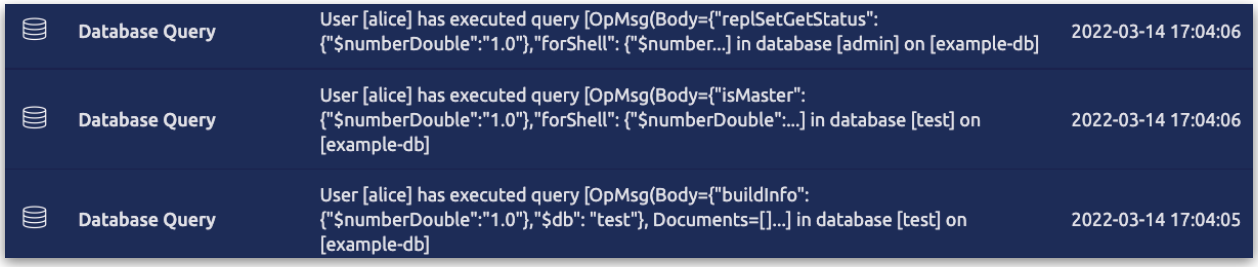
TELC-Q122-11. Database Query Audit Bypass	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	Database Access
Status	Closed

Description

Database queries using Teleport Database Access are recorded to the Audit Log.

With MongoDB, for example, this occurs inside `authorizeClientMessage` in lib/srv/db/mongodb/engine.go#197:

```
defer e.Audit.OnQuery(e.Context, sessionCtx, common.Query{
    Database: database,
    Query:    message.String(),
    Error:    err,
})
```



This adds an `AuditEvent` to the audit writer's queue. The queue is processed in the go routine in `NewAuditWriter` in lib/events/auditwriter.go#37-69. If the `AuditWriter` encounters an error writing the event, such as by trying to write more than 65535 characters, the audit writer is closed. Once the audit writer is closed, further queries for that session will also not be logged as events cannot be added to the queue.

Audit errors do not prevent queries from being proxied to the database server, so it is possible to make queries which are not audited by Teleport.

This affects Application Access also. Requests can be made without being audited, but once the audit writer is closed the application's responses are no longer returned. The user just receives Internal Server Error.

Reproduction Steps

Connect to a MongoDB database and run the following long query via `mongosh`:

```
db.runCommand({isMaster:1, saslSupportedMechs: ".".repeat(65535)})
```

The query will return successfully but nothing will appear in the Audit Log. The Teleport Database Access daemon will log the following error from lib/events/emitter.go#484:

```
ERRO            Failed to emit audit event db.session.query(TDB02I). error:[
ERROR REPORT:
Original Error: trace.aggregate record size 65956 exceeds max message size
of 65536 bytes
Stack Trace:
    /go/src/github.com/gravitational/teleport/lib/events/
emitter.go:570 github.com/gravitational/teleport/lib/events.
(*TeeStream).EmitAuditEvent
    /go/src/github.com/gravitational/teleport/lib/events/
emitter.go:482 github.com/gravitational/teleport/lib/events.
(*CheckingStream).EmitAuditEvent
    /go/src/github.com/gravitational/teleport/lib/events/
auditwriter.go:421 github.com/gravitational/teleport/lib/events.
(*AuditWriter).processEvents
    /go/src/github.com/gravitational/teleport/lib/events/
auditwriter.go:65 github.com/gravitational/teleport/lib/
events.NewAuditWriter.func1
    /opt/go/src/runtime/asm_amd64.s:1581 runtime.goexit
User Message: record size 65956 exceeds max message size of 65536 bytes] events/
emitter.go:484
```

This does not contain the query body or the username.

Further database queries of any size within the same session also return successfully without appearing in the Audit Log. However, the query body and session information will be logged by the Teleport Database Access daemon at lib/srv/db/common/audit.go#156:

```
ERRO [DB:AUDIT]  Failed to emit audit event:
Metadata:<Index:15 Type:"db.session.query" ID:"17cde9f3-6d21-46e3-a6a8-
e8d5a6ea913d" Code:"TDB02I" Time:<seconds:1647269811 nanos:735000000 >
ClusterName:"xn--fwz71v.cloud.gravitational.io" > User:<User:"alice" >
Session:<SessionID:"a3f5997e-480f-4910-8ae1-275e62e1e1c8" >
Database:<DatabaseService:"example-
db" DatabaseProtocol:"mongodb" DatabaseURI:"localhost:27017" DatabaseName:"test"
DatabaseUser:"alice" > DatabaseQuery:"OpMsg(Body={\"isMaster\":
{\"$numberDouble\":\"1.0\"},\"forShell\": {\"$numberDouble\":\"1.0\"},\"$db\":
\"test\"}, Documents=[], Flags=)" Status:<> . error:[
ERROR REPORT:
Original Error: *trace.ConnectionProblemError audit writer is closed
Stack Trace:
    /go/src/github.com/gravitational/teleport/lib/events/
auditwriter.go:296 github.com/gravitational/teleport/lib/events.
(*AuditWriter).EmitAuditEvent
    /go/src/github.com/gravitational/teleport/lib/srv/db/common/
audit.go:189 github.com/gravitational/teleport/lib/srv/db/common.
(*audit).emitAuditEvent
```

```
    /go/src/github.com/gravitational/teleport/lib/srv/db/common/
audit.go:185 github.com/gravitational/teleport/lib/srv/db/common.(*audit).OnQuery
    /go/src/github.com/gravitational/teleport/lib/srv/db/mongodb/
engine.go:187 github.com/gravitational/teleport/lib/srv/db/mongodb.
(*Engine).authorizeClientMessage
    /go/src/github.com/gravitational/teleport/lib/srv/db/mongodb/
engine.go:106 github.com/gravitational/teleport/lib/srv/db/mongodb.
(*Engine).handleClientMessage
    /go/src/github.com/gravitational/teleport/lib/srv/db/mongodb/
engine.go:86 github.com/gravitational/teleport/lib/srv/db/mongodb.
(*Engine).HandleConnection
    /go/src/github.com/gravitational/teleport/lib/srv/db/
server.go:688 github.com/gravitational/teleport/lib/srv/db.
(*Server).handleConnection
    /go/src/github.com/gravitational/teleport/lib/srv/db/
server.go:633 github.com/gravitational/teleport/lib/srv/db.
(*Server).HandleConnection
    /go/src/github.com/gravitational/teleport/lib/reversetunnel/
transport.go:275 github.com/gravitational/teleport/lib/reversetunnel.
(*transport).start
    /opt/go/src/runtime/asm_amd64.s:1581 runtime.goexit
User Message: audit writer is closed] common/audit.go:190
```

The issue with Application Access can be reproduced by fetching a long URL e.g.:

```
curl -i \
  --path-as-is \
  --cookie "__Host-grv_app_session=ec31b..." \
  https://sales-center-staging.platform.teleport.sh/$(python -c 'print("./" *
40000)')
```

The URL is printed to the Teleport Application Access daemon logs but not the Audit Log. Further requests reach the application but return Internal Server Error.

Impact

Medium. Database queries and HTTP requests to Applications can be performed without appearing in the Audit Log. The initial long database query will not be logged to the Teleport daemon log either.

Complexity

Low. Crafting a large database query requires basic skills for the relevant database. Access to the database or application via Teleport is required.

Remediation

Review error handling for audit failures.

When the `AuditWriter` closes with an error, reopen the `AuditWriter` and maybe log the username to the Audit Log. Alternatively, the entire session could be killed.
Consider truncating Audit Log messages to prevent this error.

Appendix A - Vulnerability Classification

Vulnerability Severity	**Critical**
	High
	Medium
	Low
	Informational
Vulnerability Class	Components With Known Vulnerabilities
	Covert Channel (Timing Attacks, etc.)
	Cross Site Request Forgery (CSRF)
	Cross Site Scripting (XSS)
	Denial of Service (DoS)
	Information Exposure
	Injection Flaws (SQL, XML, Command, Path, etc)
	Insecure Design
	Insecure Direct Object References (IDOR)
	Insufficient Authentication and Session Management
	Insufficient Authorization
	Insufficient Cryptography
	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
	Race Condition
	Security Misconfiguration
	Server-Side Request Forgery (SSRF)
	Unrestricted File Uploads
	Unvalidated Redirects and Forwards
	User Privacy
	Time-of-Check to Time-of-Use (TOCTOU)
	Insecure Deserialization

Appendix B - Remediation Checklist

The table below can be used to keep track of your remediation efforts inside this report. Mark the boxes when a fix has been implemented for the vulnerability.

☐	**Apply sanity-checks and error handling to RPC calls.**
☐	**Fields that begin with Equals (=), Plus (+), Minus (-), or At ("@") characters should be prepended by a single quote (') or apostrophe (') character.**
☐	**Set a bucket policy which denies access if** `aws:SecureTransport` **is false.**
☐	**Add a public access block to all buckets which should not contain any public objects.**
☐	**Enable default Server-Side Encryption on all S3 buckets.**
☐	**All provisioning, deletion, scaling, tagging and upgrading tasks should be performed exclusively by the tenant operator rather than the tenant itself. Remove the application-autoscaling access. Limit the allowed DynamoDB actions. Restrict the allowed S3 storage classes. Fix the SSRF vulnerability.**
☐	**Fix the condition inside the** `parseChangeUserPacket`**.**
☐	**Disallow the SSH web client functionality without prior authorization.**
☐	**If the WebSocket Upgrade fails, immediately return an error.**
☐	**Review error handling for audit failures.**

When done patching the listed vulnerabilities, many clients find it worthwhile to perform a retest. During a retest, Doyensec researchers will attempt to bypass and subvert all implemented fixes. Retests usually take one or two days. Please reach out if you'd like more information on our retesting process.

Appendix C - Retesting Results

Retesting was conducted by Doyensec's security researchers to ensure that the Teleport Cloud issues reported in Q1 2022 were completely addressed.

A summary of the current state is summarized below:

ID	Title	Vulnerability Class	Severity	Status
TELC-Q122-1	**Authenticated Denial of Service in Sales Center**	**Denial of Service (DoS)**	**Low**	**Closed**
Comment	• Sending an empty gRPC request is no longer causing the server to panic			
TELC-Q122-2	**CSV Injection Via S3 Data Export Using Sales Center Service**	**Injection Flaws (SQL, XML, Command, Path, etc)**	**Informational**	**Risk Accepted**
Comment	• The risk carried by the issue was accepted by the team			
TELC-Q122-3	**S3 Buckets Allow Insecure Transport**	**Security Misconfiguration**	**Low**	**Open**
Comment	• S3 actions over insecure transport are only prevented for the session bucket. See the new `deny_http_only_access` policy defined at `cloud/deploy/terraform/teleport-cloud.tf:1095`			
TELC-Q122-4	**Private S3 Buckets Without Public Access Block**	**Security Misconfiguration**	**Low**	**Closed**

Comment	• Fixed by pull request #1513 for the vulnerable buckets.
	A new S3 bucket policy (`aws_s3_bucket_public_access_block`) was added having:
	`block_public_acls = true`
	`block_public_policy = true`
	`ignore_public_acls = true`
	`restrict_public_buckets = true`			

TELC-Q122-5	**S3 Buckets With Encryption Disabled**	**Security Misconfiguration**	**Low**	**Closed**

Comment	• Buckets Server Side Encryption (SSE) added to vulnerable buckets. The following SSE with Key Management Services (KMS) was applied:
	`{`
	`  bucket_key_enabled = true`
	`  apply_server_side_encryption_by_default {`
	`  sse_algorithm = "aws:kms"`
	`  }`
	`}`			

ID	Title	Vulnerability Class	Severity	Status
TELC-Q122-6	**Tenant IAM Policy Too Broad**	**Insecure Design**	**Medium**	**Closed**
Comment	• The `tenant_role` was updated according to the least privilege principle			
TELC-Q122-7	**Wildcard Resource Permissions in Staging Role**	**Security Misconfiguration**	**Medium**	**Closed**
Comment	• A wider-than-necessary `resource` permissions from `kube-staging-access` role was removed			
TELC-Q122-8	**Denial Of Service Via Malicious MySQL COM_CHANGE_USER Packet**	**Denial of Service (DoS)**	**Low**	**Closed**
Comment	• The Teleport core team addressed a number of MySQL panics, including this issue			
TELC-Q122-9	**Teleport Allows Connection To Arbitrary SSH server**	**Insecure Design**	**Low**	**Risk Accepted**
Comment	• The issue was marked as "Risk Accepted" in 2021 & 2022, but the Teleport Core team is considering changing this behavior in the future			
TELC-Q122-10	**Denial Of Service Via WebSocket Upgrader**	**Denial of Service (DoS)**	**Medium**	**Closed**
Comment	• Teleport now returns on error instead of attempting to continue on failure to establish websocket connections			
TELC-Q122-11	**Database Query Audit Bypass**	**Insecure Design**	**Low**	**Closed**
Comment	• Audit events are now still being logged after encountering an error while trying to write an event			