SUBGRAPH

GLOBALEAKS - FINDINGS REPORT

GlobaLeaks Security Assessment

February 14, 2018

Prepared for: GlobaLeaks

Subgraph Technologies, Inc.
642 Rue de Courcelle, Suite 309
Montreal, Quebec
https://subgraph.com

Overview

In January of 2018, Subgraph performed a security audit of the GlobaLeaks open-source whistleblowing platform. We were requested to perform an audit of the new multi-tenancy implementation in the application as well as a general security assessment.

The test resulted in two minor security findings. Our overall assessment is that the application has been designed to defend against common application security issues. The multi-tenancy implementation also appears to provide an adequate level of isolation against typical attack scenarios such attempts by users of one deployed instance to access other instances.

In addition to the security findings, this report includes our observations and notes about the security measures that have been implemented.

Testing activities

To perform this audit, we ran our own local instance of GlobaLeaks in *Docker* as well as on a separate dedicated computer. Most of the testing was performed via direct network access to the service on TCP port 8082. However, some tests were also performed via the Tor hidden service '*.onion*' addresses.

Our tests consisted of the following activities:

- Automated and manual testing of the user interface (unauthenticated and under different application roles)
- Automated and manual testing of the REST API (unauthenticated and under different application roles)
- Testing the authorization of roles and tenants to ensure the following:
  - Unauthenticated users cannot gain access as an existing Whistleblower, recipient, custodian, or administrator
  - Authenticated users cannot elevate privileges
  - Users within one tenant cannot gain access to another tenant
- Code auditing of security critical parts of the application, including:
  - Authentication
  - Authorization
  - Handling of input
  - Database usage
  - Multi-tenancy support

Summary

No.	Title	Severity	Remediation
V-001	Pre-Authentication File Upload Processing	Medium	Resolved
V-002	Exception API Phishing Vulnerability	Low	Unresolved

Details

V-001: Pre-Authentication File Upload Processing

Severity	Remediation
Medium	Resolved

Discussion

Subgraph discovered a security issue in relation to how file uploads such as profile images are processed by the GlobaLeaks application. This affects all of the file upload APIs of the application.

The issue is that some pre-processing is performed prior to authentication. In particular, the application will make preparations to encrypt the uploaded files.

If the user is not authenticated, the file upload will be rejected by the application but the files that were created while preparing to encrypt the uploaded file will remain open. These open file descriptors will count towards the maximum number of open files for the process.

We noticed that the `globaleaks.init` file included in the Debian packaging configuration attempts to raise this limit. This will prevent the issue from occurring earlier. It is also worth noting that if the application is run inside of Docker, it will use the Docker limit, which defaults to *1024*. This is how we initially triggered the issue. If deploying in Docker, the global limit should be raised. Raising the limit in any case does not prevent the vulnerability, it just increases the time and effort required to trigger it.

An attacker can exploit this to cause a denial-of-service to the application by uploading a number of files that exceeds the limit allowed for the process. Once this limit is reached, the application will not be able to open any more files.

While in this state, the application will be blocked from establishing new connections. In this case, the following error message appears in the logs:

```
[twisted.web.server.Site] Could not accept new connection (EMFILE)
```

The other side-effect we observed in the logs was an inability to open the *SQLite* database file:

```
[-] OperationalError: (sqlite3.OperationalError) unable to open database
↪   file
```

The application will eventually recover if the attack stops.

Impact Analysis

Files uploaded as a result of this vulnerability are not accepted by the application. However, a denial-of-service is possible because the application still creates the AES keys that are meant to encrypt the uploaded files. It does this before authenticating the user.

The issue affects all of the file upload APIs. It is to exploitable through the submission interface once a token has been obtained. However, other interfaces including those in the `admin` path can be abused without requiring a token or session ID. This is more effective than trying to exploit the issue through the submission interface – the user does not need to submit tokens and it may generate fewer notifications than submissions do.

We were able to reproduce this condition over Tor using the *.onion* address of our deployed application.

While accessing the application by its *.onion* address is slow, we were able to able to cause the DoS by running a large number of file uploads in parallel.

The attack does not require much bandwidth – it sufficient to send a file upload without any content, resulting in a request that is approximately `1KB` in size. Running the file uploads in parallel (we used 500 *Burp Intruder* threads) is realistic from a single computer running commodity hardware. It took 30-60 minutes to reproduce with a file descriptor limit of *32768*, without trying to further optimize or distribute the attack.

The attack would be much faster on an instance running on the public Internet. However, attacking hosts could be then be blocked by source IP and the normal denial-of-service incident response measures are applicable.

As a side effect, when the denial-of-service condition has been triggered, the application will print the path and name of AES key it tried to generate in the response messages. This is not very sensitive information but the application *should* display a generic error instead of the full exception.

Remediation Recommendations

Require authentication prior to performing any other operations. The application prepares to accept a file upload prior to authenticating the user. If authentication checks are applied before these preparations are made then it is no longer possible for an unauthenticated user to exploit the issue.

The *flowIdentifier* field is used to determine whether or not to prepare the uploaded file for encryption. While this field is generated by the application, an arbitrary value will also be accepted. The application should check that the *flowIdentifier* it receives is one that it created rather than an arbitrary user-supplied value. If the supplied value was not created by the application then the request should be rejected without preparing to encrypt the file.

Complete error messages should not be displayed to the user as exception information may aid in the discovery and exploitation of vulnerabilities. The application should instead display generic error messages to users.

GlobaLeaks may want to consider adding signatures for invalid file uploads to the anomaly detection mechanism. This could include file uploads with a malformed *flowIdentifier*. File uploads to privileged APIs that are missing the 'X-Session' request header are also suspicious.

Rate-limiting, similar to what is implemented in the brute-force protections, may also help to limit exploitation of this and other issues.

Remediation Status

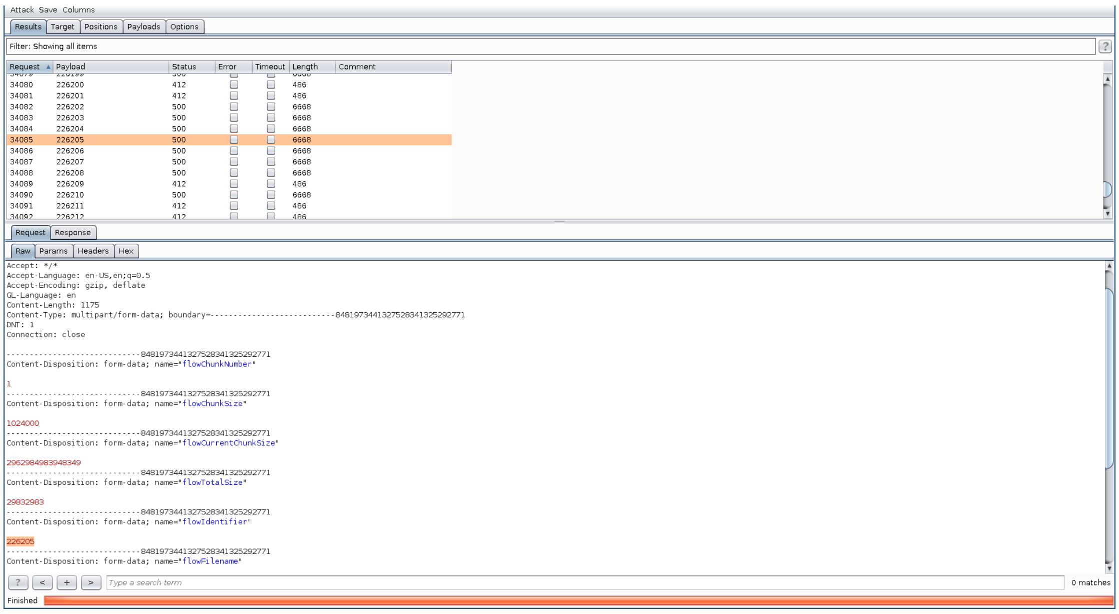
Resolved. GlobaLeaks addressed this issue in the following commit:

`https://github.com/globaleaks/GlobaLeaks/commit/f184b08a7432107297abf7e351a4dce02b513088`

Subgraph re-tested the application with the fix applied and could no longer reproduce the issue.

Additional Information

The following screenshot shows a file upload request in Burp intruder:

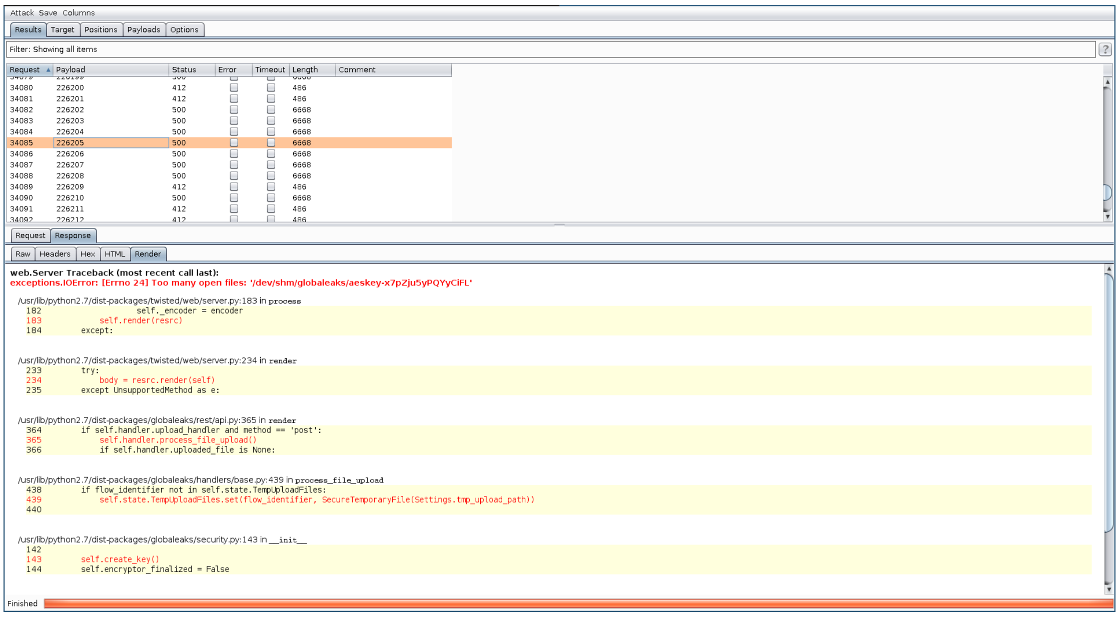

Attack Save Columns
Results Target Positions Payloads Options
Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
34079	226199	500	☐	☐	6668	
34080	226200	412	☐	☐	486	
34081	226201	412	☐	☐	486	
34082	226202	500	☐	☐	6668	
34083	226203	500	☐	☐	6668	
34084	226204	500	☐	☐	6668	
34085	226205	500	☐	☐	6668	
34086	226206	500	☐	☐	6668	
34087	226207	500	☐	☐	6668	
34088	226208	500	☐	☐	6668	
34089	226209	412	☐	☐	486	
34090	226210	500	☐	☐	6668	
34091	226211	412	☐	☐	486	
34092	226212	412	☐	☐	486	

Request Response
Raw Params Headers Hex

```
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
GL-Language: en
Content-Length: 1175
Content-Type: multipart/form-data; boundary=---------------------------8481973441327528341325292771
DNT: 1
Connection: close

-----------------------------8481973441327528341325292771
Content-Disposition: form-data; name="flowChunkNumber"

1
-----------------------------8481973441327528341325292771
Content-Disposition: form-data; name="flowChunkSize"

1024000
-----------------------------8481973441327528341325292771
Content-Disposition: form-data; name="flowCurrentChunkSize"

2962984983948349
-----------------------------8481973441327528341325292771
Content-Disposition: form-data; name="flowTotalSize"

29832983
-----------------------------8481973441327528341325292771
Content-Disposition: form-data; name="flowIdentifier"

226205
-----------------------------8481973441327528341325292771
Content-Disposition: form-data; name="flowFilename"
```

The following screenshot shows the response:



Request	Payload	Status	Error	Timeout	Length	Comment
34080	226200	412	☐	☐	486	
34081	226201	412	☐	☐	486	
34082	226202	500	☐	☐	6668	
34083	226203	500	☐	☐	6668	
34084	226204	500	☐	☐	6668	
34085	226205	500	☐	☐	6668	
34086	226206	500	☐	☐	6668	
34087	226207	500	☐	☐	6668	
34088	226208	500	☐	☐	6668	
34089	226209	412	☐	☐	486	
34090	226210	500	☐	☐	6668	
34091	226211	412	☐	☐	486	
34092	226212	412	☐	☐	486	

```
web.Server Traceback (most recent call last):
exceptions.IOError: [Errno 24] Too many open files: '/dev/shm/globaleaks/aeskey-x7pZju5yPQYyCiFL'

 /usr/lib/python2.7/dist-packages/twisted/web/server.py:183 in process
  182                   self._encoder = encoder
  183               self.render(resrc)
  184           except:

 /usr/lib/python2.7/dist-packages/twisted/web/server.py:234 in render
  233           try:
  234               body = resrc.render(self)
  235           except UnsupportedMethod as e:

 /usr/lib/python2.7/dist-packages/globaleaks/rest/api.py:365 in render
  364           if self.handler.upload_handler and method == 'post':
  365               self.handler.process_file_upload()
  366               if self.handler.uploaded_file is None:

 /usr/lib/python2.7/dist-packages/globaleaks/handlers/base.py:439 in process_file_upload
  438           if flow_identifier not in self.state.TempUploadFiles:
  439               self.state.TempUploadFiles.set(flow_identifier, SecureTemporaryFile(Settings.tmp_upload_path))
  440

 /usr/lib/python2.7/dist-packages/globaleaks/security.py:143 in __init__
  142
  143           self.create_key()
  144           self.encryptor_finalized = False
```

V-002: Exception API Phishing Vulnerability

Severity	Remediation
Low	Unresolved

Discussion

The application allows unauthenticated access to the exception REST API endpoint. This API will send exception emails to the users configured to receive notifications.

The exception emails are generated based on information supplied to the API by the user. This lets users inject potentially misleading or malicious content into exception emails.

The handler for the exception API allows unauthenticated access by default (the `check_roles` directive is set to a wildcard value).

Impact Analysis

This issue can be exploited to launch phishing attacks against users configured to receive exception notification emails. As they will appear to come from the application, and may even be encrypted if encrypted notifications are configured, the target of the phishing attack may trust the email contents. The emails will still include exception information so in all likelihood an attacker would exploit this to trick a user into visiting a malicious link or provide misleading information about the state of the application.

The `errorUrl` field of the exception request allows unformatted input to be injected in the notification emails.

Remediation Recommendations

Our understanding of this API is that it is meant to capture client-side exceptions. Therefore, it may be by-design that exceptions generated by unauthenticated users are captured. This makes the issue trickier to address.

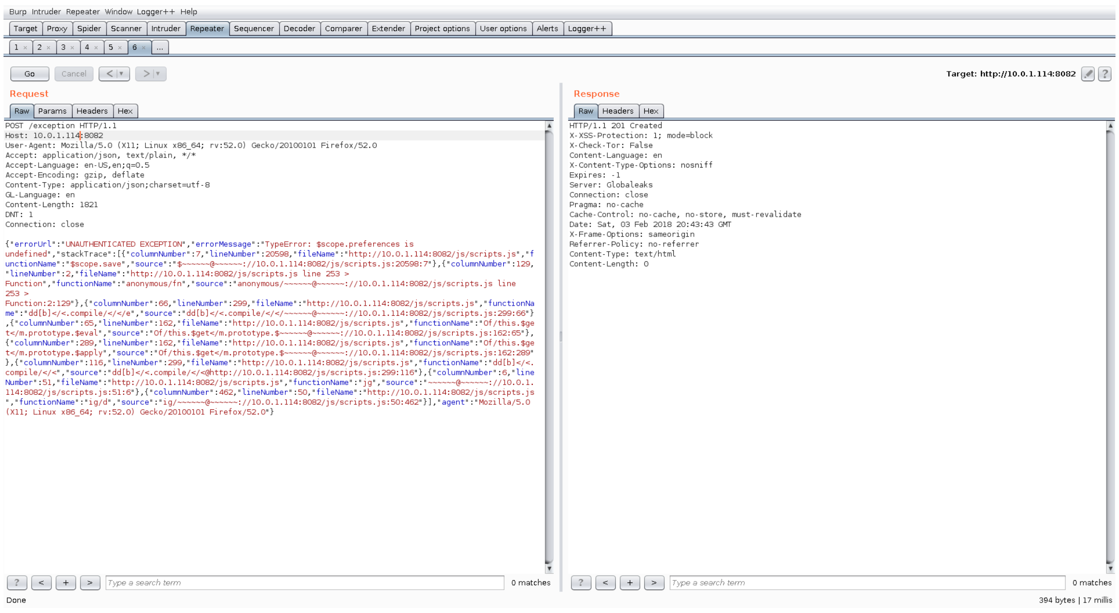
A compromise may be to only enable unauthenticated exceptions to be generated in *development* mode and to require authentication in *production* mode.

GlobaLeaks could also try to filter or structure the exception information in such a way that it cannot be easily manipulated by the user. This may also be challenging to implement.

Another option is to advise the recipient of the exception email that the report may contain untrusted information delivered from the client.

Additional Information

The following screenshot shows an exception request from an unauthenticated user that contains user-supplied input:



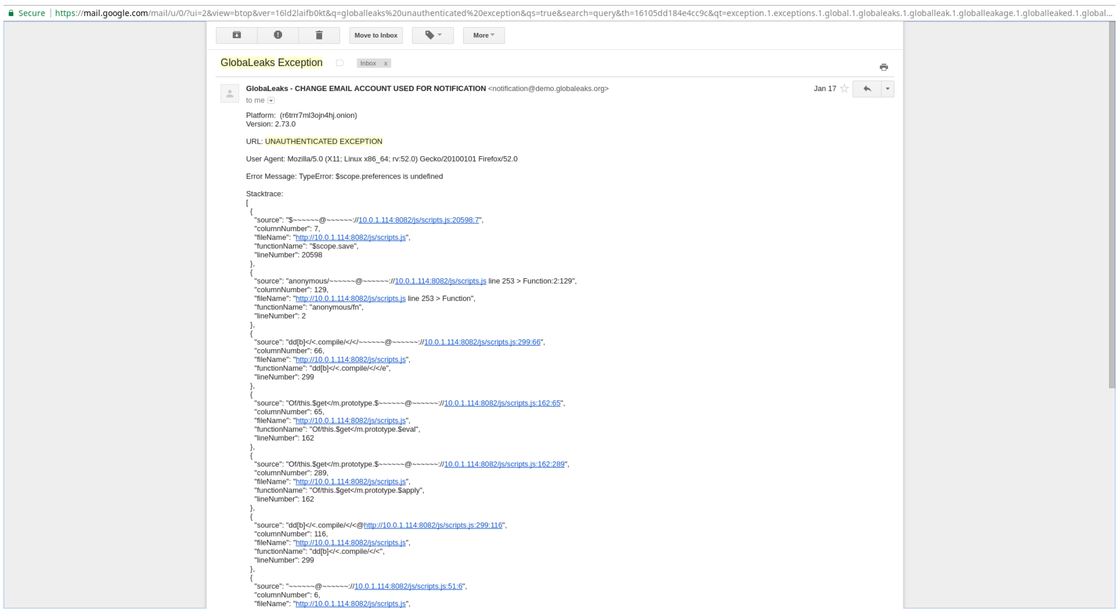
```
POST /exception HTTP/1.1
Host: 10.0.1.114:8082
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json;charset=utf-8
GL-Language: en
Content-Length: 1821
DNT: 1
Connection: close

{"errorUrl":"UNAUTHENTICATED EXCEPTION","errorMessage":"TypeError: $scope.preferences is
undefined","stackTrace":[{"columnNumber":7,"lineNumber":20598,"fileName":"http://10.0.1.114:8082/js/scripts.js","f
unctionName":"$scope.save","source":"$~~~~~~@~~~~~~://10.0.1.114:8082/js/scripts.js:20598:7"},{"columnNumber":129,
"lineNumber":2,"fileName":"http://10.0.1.114:8082/js/scripts.js line 253 >
Function","functionName":"anonymous/fn","source":"anonymous/~~~~~~@~~~~~~://10.0.1.114:8082/js/scripts.js line
253 >
Function:2:129"},{"columnNumber":66,"lineNumber":299,"fileName":"http://10.0.1.114:8082/js/scripts.js","functionNa
me":"dd[b]</<.compile/</</e","source":"dd[b]</<.compile/</</~~~~~~@~~~~~~://10.0.1.114:8082/js/scripts.js:299:66"}
,{"columnNumber":65,"lineNumber":162,"fileName":"http://10.0.1.114:8082/js/scripts.js","functionName":"Of/this.$ge
t</m.prototype.$eval","source":"Of/this.$get</m.prototype.$~~~~~~@~~~~~~://10.0.1.114:8082/js/scripts.js:162:65"},
{"columnNumber":289,"lineNumber":162,"fileName":"http://10.0.1.114:8082/js/scripts.js","functionName":"Of/this.$ge
t</m.prototype.$apply","source":"Of/this.$get</m.prototype.$~~~~~~@~~~~~~://10.0.1.114:8082/js/scripts.js:162:289"
},{"columnNumber":116,"lineNumber":299,"fileName":"http://10.0.1.114:8082/js/scripts.js","functionName":"dd[b]</<.
compile/</<","source":"dd[b]</<.compile/</<@http://10.0.1.114:8082/js/scripts.js:299:116"},{"columnNumber":6,"line
Number":51,"fileName":"http://10.0.1.114:8082/js/scripts.js","functionName":"jg","source":"~~~~~~@~~~~~~://10.0.1.
114:8082/js/scripts.js:51:6"},{"columnNumber":462,"lineNumber":50,"fileName":"http://10.0.1.114:8082/js/scripts.js
","functionName":"ig/d","source":"ig/~~~~~~@~~~~~~://10.0.1.114:8082/js/scripts.js:50:462"}],"agent":"Mozilla/5.0
(X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0"}
```

```
HTTP/1.1 201 Created
X-XSS-Protection: 1; mode=block
X-Check-Tor: False
Content-Language: en
X-Content-Type-Options: nosniff
Expires: -1
Server: Globaleaks
Connection: close
Pragma: no-cache
Cache-Control: no-cache, no-store, must-revalidate
Date: Sat, 03 Feb 2018 20:43:43 GMT
X-Frame-Options: sameorigin
Referrer-Policy: no-referrer
Content-Type: text/html
Content-Length: 0
```

Target: http://10.0.1.114:8082

394 bytes | 17 millis

The following screenshot shows the exception email with the injected content:



GlobaLeaks Exception

GlobaLeaks - CHANGE EMAIL ACCOUNT USED FOR NOTIFICATION <notification@demo.globaleaks.org>
to me

Jan 17

```
Platform:  (r6trrr7ml3ojn4hj.onion)
Version: 2.73.0

URL: UNAUTHENTICATED EXCEPTION

User Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0

Error Message: TypeError: $scope.preferences is undefined

Stacktrace:
[
  {
    "source": "$~~~~~~@~~~~~~://10.0.1.114:8082/js/scripts.js:20598:7",
    "columnNumber": 7,
    "fileName": "http://10.0.1.114:8082/js/scripts.js",
    "functionName": "$scope.save",
    "lineNumber": 20598
  },
  {
    "source": "anonymous/~~~~~~@~~~~~~://10.0.1.114:8082/js/scripts.js line 253 > Function:2:129",
    "columnNumber": 129,
    "fileName": "http://10.0.1.114:8082/js/scripts.js line 253 > Function",
    "functionName": "anonymous/fn",
    "lineNumber": 2
  },
  {
    "source": "dd[b]</<.compile/</</~~~~~~@~~~~~~://10.0.1.114:8082/js/scripts.js:299:66",
    "columnNumber": 66,
    "fileName": "http://10.0.1.114:8082/js/scripts.js",
    "functionName": "dd[b]</<.compile/</</e",
    "lineNumber": 299
  },
  {
    "source": "Of/this.$get</m.prototype.$~~~~~~@~~~~~~://10.0.1.114:8082/js/scripts.js:162:65",
    "columnNumber": 65,
    "fileName": "http://10.0.1.114:8082/js/scripts.js",
    "functionName": "Of/this.$get</m.prototype.$eval",
    "lineNumber": 162
  },
  {
    "source": "Of/this.$get</m.prototype.$~~~~~~@~~~~~~://10.0.1.114:8082/js/scripts.js:162:289",
    "columnNumber": 289,
    "fileName": "http://10.0.1.114:8082/js/scripts.js",
    "functionName": "Of/this.$get</m.prototype.$apply",
    "lineNumber": 162
  },
  {
    "source": "dd[b]</<.compile/</<@http://10.0.1.114:8082/js/scripts.js:299:116",
    "columnNumber": 116,
    "fileName": "http://10.0.1.114:8082/js/scripts.js",
    "functionName": "dd[b]</<.compile/</<",
    "lineNumber": 299
  },
  {
    "source": "~~~~~~@~~~~~~://10.0.1.114:8082/js/scripts.js:51:6",
    "columnNumber": 6,
    "fileName": "http://10.0.1.114:8082/js/scripts.js",
```

Observations

This section includes the observations and notes we gathered during the tests, code audit, and through reading the GlobaLeaks software security documentation. It is not intended to cover every aspect of the security design but rather highlight how some security features have been implemented and provide suggestions.

Roles

The GlobaLeaks application includes a number of different roles that are granted privileges according to the types of operations they are expected to perform. The following roles are supported:

- Admin (administers the application, each tenant has a different admin account)
- Custodian
- Recipient (accepts submissions, interacts with whistleblowers)
- Whistleblower (creates submissions, interacts with recipients)

The handlers implements for each API implement the privileges for these roles. A `check_roles` directive is added to the constructor for each handler. The directive is then enforced by an authentication decorator that is implemented by the base handler class that all other handlers inherit from.

We performed tests to attempt to bypass the role enforcement mechanism but did not find any security issues with it. While the role mechanism appears to function correctly, care must be taken when assigned roles to any handler. In finding **V-002** we describe an issue where unauthenticated users are able to send input to the exception handler. This is because the `check_roles` directive is set to a wildcard (`*`).

Submissions and flood protection

GlobaLeaks tries to mitigate the risks of accepting anonymous submissions over Tor, such as denial-of-service attacks. GlobaLeaks has performed an analysis of these concerns and implemented some controls to address the problem. Some of these measures are described below. Further information can also be found within their code repository:

https://github.com/globaleaks/GlobaLeaks/blob/master/backend/doc/floodprotection.md

The application implements a temporary session for new submissions. When creating a new submission, a token will be issued and assigned a number of remaining uses (*30*). Each request will decrement the counter and when it reaches zero, a new token must be obtained. While this adds some complexity for an attacker, it can still be scripted.

To compliment this, a CAPTCHA mechanism is activated once certain anomaly thresholds have been passed. This is not a strong protection and can be trivially bypassed. The CAPTCHA is a simple math equation that can be extracted and automatically calculated by a script.

A motivated attacker could script the entire submission process. The script including a solver for CAPTCHAs when prompts are seen. Automation of submissions may be a nuisance for the administrators and recipients or could result in a denial-of-service.

Stronger protections may prevent this but must be implemented carefully as to not introduce risks to legitimate whistleblowers. For example, use of third-party services to provide stronger CAPTCHAs is likely not a desirable trade-off because it means that the whistleblower's browser must communicate with third-parties.

The anomaly detection features will provide notifications of this type of abusive behavior. Limits are also placed on the maximum size of uploaded files.

Rate-limiting, similar to the brute-force protection, may help to limit automation.

In any case, this seems to be an open problem with no easy solution. This is why we have not highlighted it as a security finding in the report. The developers have researched the problem and implemented some counter-measures that are not without trade-offs. All such counter-measures must be carefully evaluated to ensure that they do not introduce risks to the whistleblower, the application, or add more potential for denial-of-service attacks.

Multi-tenancy implementation

GlobaLeaks implemented multi-tenacy in the application. The use-case for multi-tenancy is to enable multiple organizations (or branches within the same organization) to accept submissions to different endpoints within a single deployment of the application.

The typical multi-tenant deployment of GlobaLeaks would involve different organizations or branches of an organization that may theoretically share submissions with each other. Due to this, the security boundary between tenant access to submissions was not as important as other concerns. The multi-tenant implementation was still designed to prevent this type of cross-tenant access.

Each tenant is assigned a number ID. When handling requests, the application must determine the destination tenant for the request. In this case, the tenant ID is either the first one or it is calculated from the provided HTTP `Host` header (which, for example, may be the `.onion` address of the tenant). If an invalid `Host` is supplied, the application will fail closed (sending an error message to the user).

We performed an audit of all of the roles for each tenant to ensure that a user from one tenant could not gain access to the user of another tenant. We did not find any security issues. Each issue is assigned a tenant ID in the database and we did not find any means to bypass this control.

Web application security measures

During the audit, we observed a number of explicit security measures intended to prevent common web application vulnerabilities. There are also a number of other design choices which help to implicitly prevent these types of vulnerabilities.

Brute-forcing of credentials

GlobaLeaks has implemented counter-measures to limit brute-force credential guessing attacks. This is especially a concern when the application is accessed anonymously as a Tor Onion service.

These counter-measures include the following:

- A response delay is introduced after failed authentication attempts
- Failed authentication attempts raise the alert level, which may trigger other counter-measures and will send notifications to the administrator (which are limited to avoid spamming the administrator)

We encountered the response delays during our automated tests as they will be applied any time a large number of requests are sent to the authentication API. They slowed during our scanning of this API considerably.

The notifications may be sent as a response to other anomalies or suspicious behavior. As mentioned in the other reports, the implementation of a response delay for other types of suspicious activity could complicate other attacks such as the issue described in finding **V-001**.

Cross-site Request Forgery

The REST API does not use authentication cookies but instead authenticates users via a session ID that is sent with a custom header ('X-Session'). This behaves similarly to a cross-request forgery token as it is a value embedded within the client content that cannot be guessed by the attacker.

Directory traversal

Safe handling of file uploads and downloads is paramount to the security of the application. It is important that the application accepts file uploads safely, without posing a risk to the host or other users of the application (recipients, administrators, etc.). Likewise, as the submissions are sensitive in nature, the application should prevent unauthorized access to them.

Directory traversal vulnerabilities may allow unauthorized access to files outside the intended location of file upload or download operations. In the case of file downloads, this could unauthorized access to submissions or other sensitive files on the host. In the case of file uploads, malicious file uploads could overwrite other files on the host (with the permissions of the application), which could lead to denial of service or code execution. GlobaLeaks implements some explicit security measures to prevent this type of vulnerability.

The first security measure that the application randomly generates the filenames of any uploaded files instead of accepting externally supplied filenames from the uploader. This means that the uploaded can not influence the location of uploaded files. Instead they will be stored in a path specified by the application, using a random filename that is stored in the database. When uploaded files are requested, the application queries the database for the filenames instead of serving them directly from the host filesystem.

There is a secondary security measure to ensure that files are only served from paths that are configured in the application rather than arbitrary filesystem paths. This measure is implemented for file downloads, uploads, and the serving of static files.

```
def directory_traversal_check(trusted_absolute_prefix, untrusted_path):
    """
    check that an 'untrusted_path' matches a 'trusted_absolute_path'
 ↪  prefix
    """
    if not os.path.isabs(trusted_absolute_prefix):
        raise Exception("programming error: trusted_absolute_prefix is
         ↪  not an absolute path: %s" %
                        trusted_absolute_prefix)

    untrusted_path = os.path.abspath(untrusted_path)

    if trusted_absolute_prefix !=
    ↪  os.path.commonprefix([trusted_absolute_prefix, untrusted_path]):
        log.err("Blocked file operation for: (prefix, attempted_path) :
 ↪  ('%s', '%s')",
                trusted_absolute_prefix, untrusted_path)

        raise errors.DirectoryTraversalError
         ↪
```

As the application already detects directory traversal attempts and alerts on anomalies, GlobaLeaks may want to consider flagging directory traversal attempts as an anomalies.

Cross-site scripting and HTML injection

Protection against these attacks rely on the filtering features of the *Angular.js* framework. All input is rendered to browsers is passed through the *$sanitize* filter of the framework.

We did not discover any security issues with this approach. However, it should be noted that *Angular.js* has been subject to cross-site scripting and other vulnerabilities in the past. As with any third-party component, it is important to ensure that the framework and its dependencies are kept up-to-date.

SQL injection

GlobaLeaks stores persistent data in an *SQLite* database using the *SQLAlchemy* library.

When used according to best practices, *SQLAlchemy* parameterizes and escapes parameters supplied in queries. This provides protection against SQL injection vulnerabilities. However, if this behavior is bypassed to construct a raw query that includes unescaped user-supplied input, then this will result in a vulnerability. During our tests and code audit, we did not discover any instances of this.

Appendix

Methodology

Our approach to testing is designed to understand the design, behavior, and security considerations of the assets being tested. This helps us to achieve the best coverage over the duration of the test.

To accomplish this, Subgraph employs automated, manual and custom testing methods. We conduct our automated tests using the industry standard security tools. This may include using multiple tools to test for the same types of issues. We perform manual tests in cases where the automated tools are not adequate or reveal behavior that must be tested manually. Where required, we also develop custom tools to perform tests or reproduce test findings.

The goals of our testing methodology are to:

- Understand the expected behavior and business logic of the assets being tested
- Map out the attack surface
- Understand how authentication, authorization, and other security controls are implemented
- Test for flaws in the security controls based on our understanding
- Test every point of input against a large number of variables and observe the resulting behavior
- Reproduce and re-test findings
- Gather enough supporting information about findings to enable us to classify, report, and suggest remediations

Description of testing activities

Depending on the type and scope of the engagement, our methodology may include any of the following testing activities:

1. **Information Gathering:** Information will be gathered from publicly availble sources to help increase the success of attacks or discover new vulnerabilities
2. **Network discovery:** The networks in scope will be scanned for active, reachable hosts that could be vulnerable to compromise
3. **Host Vulnerability Assessment:** Hosts applications and services will be assessed for known or possible vulnerabilities
4. **Application Exploration:** The application will be explored using manual and automated methods to better understand the attack surface and expected behavior
5. **Session Management:** Session management in web applications will be tested for security flaws that may allow unauthorized access
6. **Authentication System Review:** The authentication system will be reviewed to determine if it can be bypassed

7. **Privilege Escalation:** Privilege escalation checks will be performed to determine if it is possible for an authenticated user to gain access to the privileges assigned to another role or administrator
8. **Input Validation:** Input validation tests will be performed on all endpoints and fields within scope, including tests for injection vulnerabilities (SQL injection, cross-site scripting, command injection, etc.)
9. **Business Logic Review:** Business logic will be reviewed, including attempts to subvert the intended design to cause unexpected behavior or bypass security controls

Reporting

Findings reports are peer-reviewed within Subgraph to produce the highest quality findings. The report includes an itemized list of findings, classified by their severity and remediation status.

Severity ratings

Severity ratings are a metric to help organizations prioritize security findings. The severity ratings we provide are simple by design so that at a high-level they can be understood by different audiences. In lieu of a complex rating system, we quantify the various factors and considerations in the body of the security findings. For example, if there are mitigating factors that would reduce the severity of a vulnerability, the finding will include a description of those mitigations and our reasoning for adjusting the rating.

At an organization's request, we will also provide third-party ratings and classifications. For example, we can analyze the findings to produce *Common Vulnerability Scoring System* (CVSS)[1] scores or *OWASP Top 10*[2] classifications.

The following is a list of the severity ratings we use with some example impacts:

Critical

Exploitation could compromise hosts or highly sensitive information

High

Exploitation could compromise the application or moderately sensitive information

Medium

Exploitation compromises multiple security properties (confidentiality, integrity, or availability)

[1] https://www.first.org/cvss/
[2] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Low

Exploitation compromises a single security property (confidentiality, integrity, or availability)

Informational

Finding does not pose a security risk or represents suspicious behavior that merits further investigation

The severity of a finding is often a product of the impact to general security properties of an application, host, network, or other information system.

These are:

Property	Description
Compromise of confidientiality	Exploitation results in authorized access to data
Compromise of integrity	Exploitation results in the unauthorized modification of data or state
Compromise of availability	Exploitation results in a degradation of performance or an inability to access resources

The actual severity of a finding may be higher or lower depending on a number of other factors that may mitigate or exacerbate it. These include the context of the finding in relation to the organization as well as the likelihood of exploitation. These are described in further detail below.

Contextual factors

Confidentiality, integrity, and availability are one dimension of the potential risk of a security finding. In some cases, we must also consider contextual factors that are unique to the organization and the assets tested.

The following is a list of those factors:

Factor	Description
Financial compromise	Exploitation may result in financial losses
Reputation compromise	Exploitation may result in damage to the reputation of the organization
Regulatory liability	Exploitation may expose the organization to regulatory liability (e.g. place them in a state of non-compliance)
Organizational impact	Exploitation may disrupt the operations of the organization

Likelihood

Likelihood measures how probable it is that an attacker exploit a finding.

This is determined by numerous factors, the most influential of which are listed below:

Likelihood	Description
Authentication required	Whether or not the attack must be authenticated
Privilege level	Whether or not an authenticated attacker requires special privileges
Public exploit	Whether or not exploit code is publicly available
Public knowledge	Whether or not the finding is publicly known
Exploit complexity	How complex it is for a skilled attacker to exploit the finding
Local vs. remote	Whether or not the finding is exposed to the network
Accessibility	Whether or not the affected asset is exposed on the public Internet
Discoverability	How easy it is for the finding to be discovered by an attacker
Dependencies	Whether or not exploitation is dependant on other findings such as information leaks

Remediation status

As part of our reporting, remediation recommendations are provided to the client. To help track the issues, we also provide a remediation status rating in the findings report.

In some cases, the organization may be confident to remediate the issue and test it internally. In other cases, Subgraph works with the organization to re-test the findings, resulting in a subsequent report reflecting remediation status updates.

If requested to re-test findings, we determine the remediation status based on our ability to reproduce the finding. This is based on our understanding of the finding and our awareness of potential variants at that time. To reproduce the results, the re-test environment should be as close to the original test environment as possible.

Security findings are often due to unexpected or unanticipated behavior that is not always understood by the testers or the developers. Therefore, it is possible that a finding or variations of the finding may still be present even if it is not reproducible during a re-test. While we will do our best to work with the organization to avoid this, it is still possible.

The findings report includes the following remediation status information:

Resolved

Finding is believed to be remediated, we can no longer reproduce it

In progress

Finding is in the process of being remediated

Unresolved

Finding is not remediated – may indicate the initial report, a finding under investigation, or one that the organization has chosen not to address

Not applicable

There is nothing to resolve, this may be the case with informational findings