Radically Open Security

Penetration Test Report

Goupile

V 1.0
Amsterdam, October 30th, 2025
Confidential

Document Properties

Client	Goupile
Title	Penetration Test Report
Target	• Goupile, a free eCRF design tool
Version	1.0
Pentester	Andrea Jegher
Authors	Andrea Jegher, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	October 1st, 2025	Andrea Jegher	Initial draft
0.2	October 14th, 2025	Marcus Bointon	Review
1.0	October 30th, 2025	Marcus Bointon	1.0

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between September 24, 2025 and October 3, 2025, Radically Open Security B.V. carried out a penetration test for Goupile.

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of Work

The scope of the penetration test was limited to the following target(s):

- Goupile, a free eCRF design tool

The scoped services are broken down as follows:

- Goupile assement: 4 days
- Report writing: 1 days
- **Total effort: 5 days**

1.3 Project Objectives

ROS will perform a code audit of Goupile with its developer in order to assess its security. To do so ROS will access the source code and guide Goupile in attempting to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

1.4 Timeline

The security audit took place between September 24, 2025 and October 3, 2025.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 Elevated, 1 Moderate and 3 Low-severity issues.

A malformed JSON key containing an exceedingly large integer value triggers a raw number handler that writes the value `0` to an attacker-controlled memory address. If that address is not accessible, the server crashes CLN-002 (page 12).

Guest users are able to list and read files from projects that are not configured to allow guest access CLN-010 (page 13). The code paths that enforce guest-access checks are only exercised when a `version` query parameter is present.

The Twilio SMS endpoint encodes the message body but fails to encode the `to` field. An attacker who can trigger a message send can inject URL query separators (e.g., `&`) into the `to` parameter CLN-011 (page 18).

User input that is reflected in server logs is not sanitized CLN-006 (page 17). By sending specially crafted query parameters an attacker can inject arbitrary lines into the log files, potentially obfuscating other log entries or triggering log-based alerts.

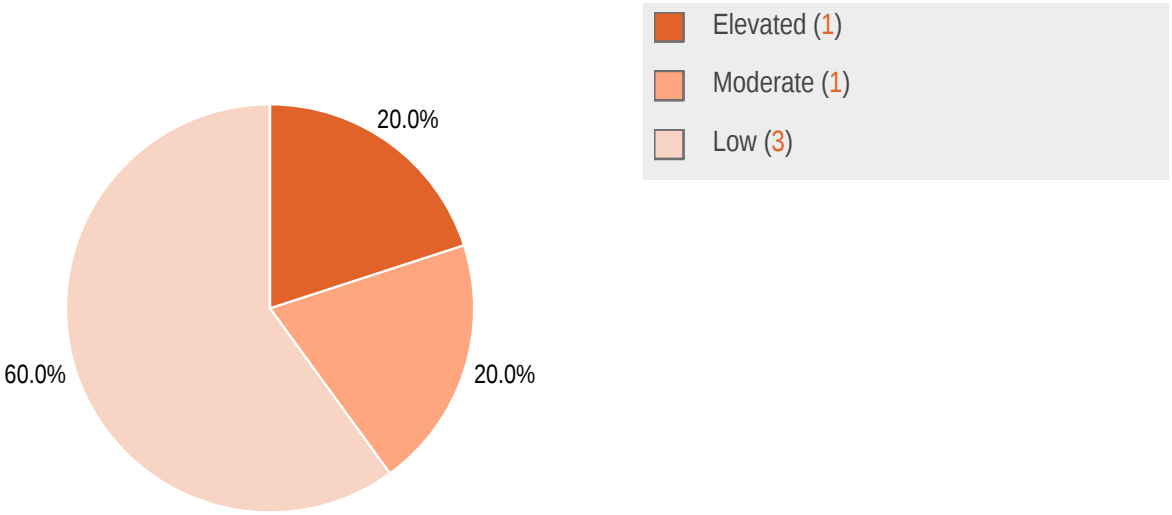
The lack of HTTP security headers could impact the security of the server since some common attacks are not mitigated CLN-012 (page 20).

1.6 Summary of Findings

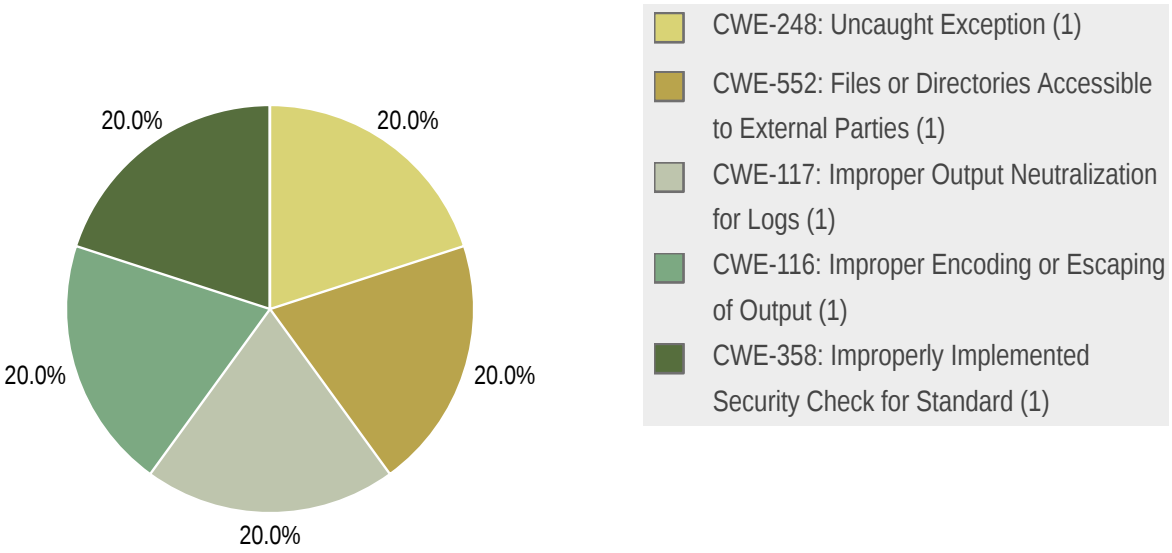
Info	Description
CLN-002 **Elevated** **Type:** CWE-248: Uncaught Exception **Status:** none	A JSON key with a long integer value will trigger a write to an attacker-controlled address of the value zero
CLN-010 **Moderate** **Type:** CWE-552: Files or Directories Accessible to External Parties **Status:** none	HandleFileList and HandleFileGet allow guest users to list and read files belonging to projects that do not allow guest users.
CLN-006 **Low** **Type:** CWE-117: Improper Output Neutralization for Logs **Status:** none	The server constructs a log message from user input, but it does not neutralize potentially dangerous elements before writing it to standard output.
CLN-011 **Low** **Type:** CWE-116: Improper Encoding or Escaping of Output **Status:** none	SendTwilio does not use FmtUrlSafe to encode the to field.

CLN-012 **Low** **Type:** CWE-358: Improperly Implemented Security Check for Standard **Status:** none	The application does not set three common security headers on the root path, these headers mitigate against common attacks on web application

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

Info	Recommendation
CLN-002 **Elevated** **Type:** CWE-248: Uncaught Exception **Status:** none	• Allocate the correct size for the integer in `json_Parser::Handler::RawNumber`.
CLN-010 **Moderate** **Type:** CWE-552: Files or Directories Accessible to External Parties **Status:** none	• Do not allow guests to access resources that are not set to allow it. • `HandleFileList` and `HandleFileGet` should check whether the instance allows guest access if there is no current user session.
CLN-006 **Low** **Type:** CWE-117: Improper Output Neutralization for Logs **Status:** none	• Escape, encode, or remove special characters like new lines in user input concatenated with logs.
CLN-011 **Low** **Type:** CWE-116: Improper Encoding or Escaping of Output **Status:** none	• Encode the `to` field using `FmtUrlSafe`.

CLN-012 **Low** **Type:** CWE-358: Improperly Implemented Security Check for Standard **Status:** none	• Implement the recommended headers. • Move inline JavaScript into external files.

2 Methodology

2.1 Planning

Our general approach during penetration tests is as follows:

1. **Reconnaissance**
   We attempt to gather as much information as possible about the target. Reconnaissance can take two forms: active and passive. A passive attack is always the best starting point as this would normally defeat intrusion detection systems and other forms of protection afforded to the app or network. This usually involves trying to discover publicly available information by visiting websites, newsgroups, etc. An active form would be more intrusive, could possibly show up in audit logs and might take the form of a social engineering type of attack.
2. **Enumeration**
   We use various fingerprinting tools to determine what hosts are visible on the target network and, more importantly, try to ascertain what services and operating systems they are running. Visible services are researched further to tailor subsequent tests to match.
3. **Scanning**
   Vulnerability scanners are used to scan all discovered hosts for known vulnerabilities or weaknesses. The results are analyzed to determine if there are any vulnerabilities that could be exploited to gain access or enhance privileges to target hosts.
4. **Obtaining Access**
   We use the results of the scans to assist in attempting to obtain access to target systems and services, or to escalate privileges where access has been obtained (either legitimately though provided credentials, or via vulnerabilities). This may be done surreptitiously (for example to try to evade intrusion detection systems or rate limits) or by more aggressive brute-force methods. This step also consist of manually testing the application against the latest (2021) list of OWASP Top 10 risks. The discovered vulnerabilities from scanning and manual testing are moreover used to further elevate access on the application.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.

- **High**

  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**

  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**

  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**

  Low risk of security controls being compromised with measurable negative impacts as a result.

3 Reconnaissance and Fingerprinting

We were able to gain information about the software and infrastructure through the following automated scans. Any relevant scan output will be referred to in the findings.

- AFLPlusPlus – https://github.com/AFLplusplus/AFLplusplus
- CodeQL – https://codeql.github.com
- Semgrep – https://github.com/semgrep/semgrep

4 Findings

We have identified the following issues:

4.1 CLN-002 — JSON parsing buffer overflow

Vulnerability ID: CLN-002

Vulnerability type: CWE-248: Uncaught Exception

Threat level: Elevated

Description:

A JSON key with a long integer value will trigger a write to an attacker-controlled address of the value zero

Technical description:

This issue was found by the fuzzing the JSON parser; the setup is shown in non-finding NF-005 (page 22).

The function that triggers the segfault is `K::json_Parser::Handler::RawNumber` at `src/core/wrap/json.cc` line 104.

Here are the steps to reproduce.

The Goupile domain must have a project named `sample-project` (or anything else, but the name must match in the request too), and it must have guest user access enabled (it works with logged-in users too, but this way there are fewer HTTP headers to show in these examples). Then it is sufficient to send the following HTTP request:

```
POST /sample-project/api/records/save HTTP/1.1
Host: 127.0.0.1:8889
Content-Length: 180
X-Requested-With: XMLHTTPRequest

{"tid":4444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
  }
```

This JSON body will trigger the segmentation fault:

```
{"tid":4444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
```

Here instead is a `curl` command to send the HTTP request:

```
curl --path-as-is -i -s -k -X $'POST' \
    -H $'Host: 127.0.0.1:8889' -H $'Content-Length: 180' -H $'X-Requested-With: XMLHTTPRequest' \
    --data-binary $'{\"tid
\":44444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
  }' \
```

```
    $'http://127.0.0.1:8889/sample-project/api/records/save'
```

Impact:

An attacker can write zero to an arbitrary address in memory.

If the address can not be accessed, it will trigger a segmentation fault, crashing the server, causing a denial of service.

Recommendation:

- Allocate the correct size for the integer in `json_Parser::Handler::RawNumber`.

4.2 CLN-010 — HandleFileList and HandleFileGet permission issue

Vulnerability ID: CLN-010

Vulnerability type: CWE-552: Files or Directories Accessible to External Parties

Threat level: Moderate

Description:

`HandleFileList` and `HandleFileGet` allow guest users to list and read files belonging to projects that do not allow guest users.

Technical description:

Function `HandleFileList` does not do any type of session check if there is no `version` URL parameter. If this parameter is missing, it will use the latest version.

```
void HandleFileList(http_IO *io, InstanceHolder *instance)
{
    const http_RequestInfo &request = io->Request();

    if (instance->master != instance) {
        LogError("Cannot list files through slave instance");
        io->SendError(403);
        return;
    }

    int64_t fs_version;
    if (const char *str = request.GetQueryValue("version"); str) {
        if (!ParseInt(str, &fs_version)) {
            io->SendError(422);
            return;
```

```
        }

        if (fs_version == 0) {
            RetainPtr<const SessionInfo> session = GetNormalSession(io, instance);

            if (!session || !session->HasPermission(instance, UserPermission::BuildCode)) {
                LogError("You cannot access pages in development");
                io->SendError(403);
                return;
            }
        }
    } else {
        fs_version = instance->fs_version.load(std::memory_order_relaxed);
    }
...
```

Similarly, `HandleFileGet` will check for permissions only if the `version` URL parameter is set to `0`:

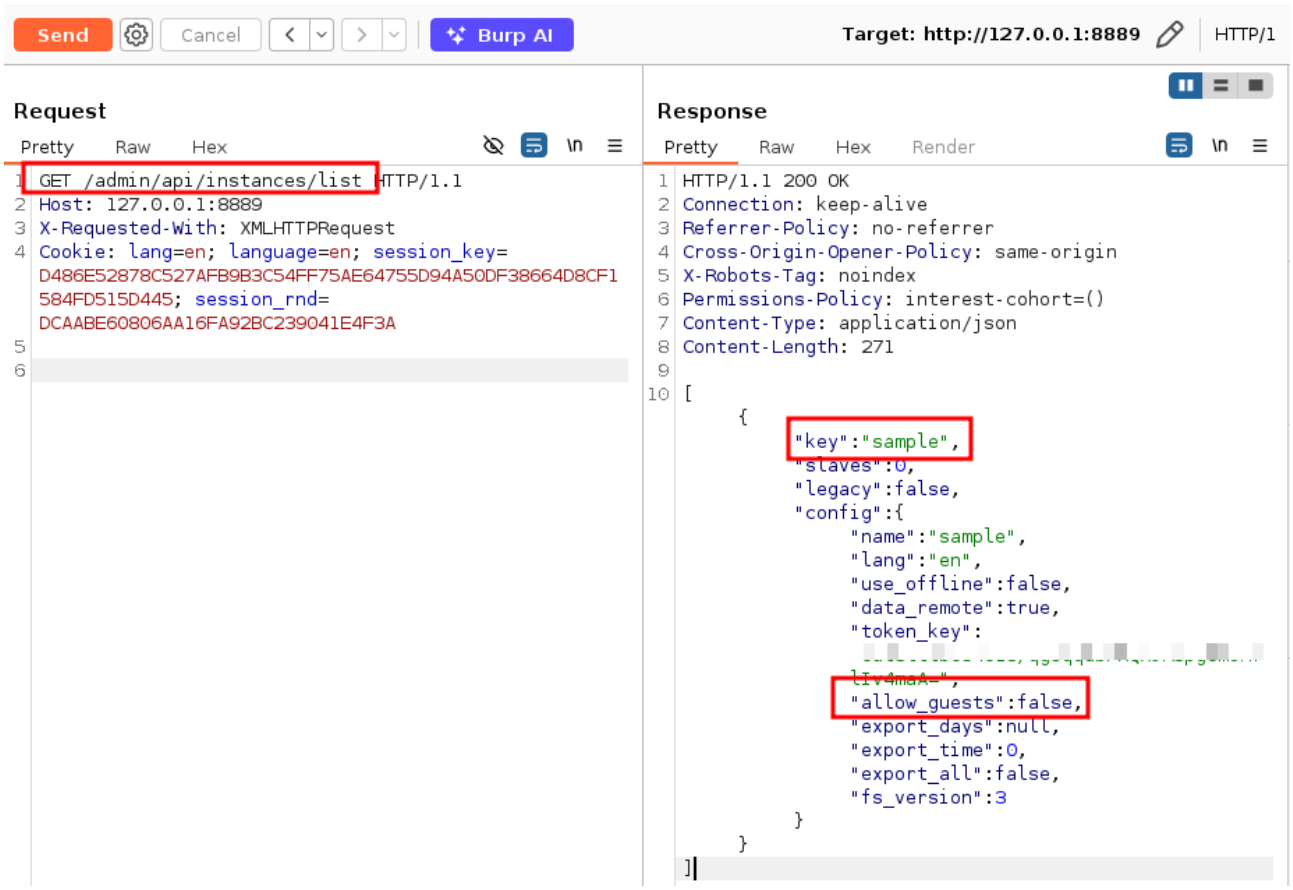
```
bool explicit_version;
    {
        Span<const char> remain;

        if (ParseInt(filename, &fs_version, 0, &remain) && remain.ptr[0] == '/') {
            if (fs_version == 0) {
                RetainPtr<const SessionInfo> session = GetNormalSession(io, instance);

                if (!session || !session->HasPermission(instance, UserPermission::BuildCode)) {
                    LogError("You cannot access pages in development");
                    io->SendError(403);
                    return true;
                }
            }

            filename = remain.Take(1, remain.len - 1);
            explicit_version = true;
        } else {
            fs_version = instance->fs_version.load(std::memory_order_relaxed);
            explicit_version = false;
        }
    }
```

To reproduce this issue, create a new project with the demo files, or choose any existing project without guest user access (in this case one named "sample"). Then perform the following requests. In the following image we see that `allow_guest` is set to `false` in the response body:

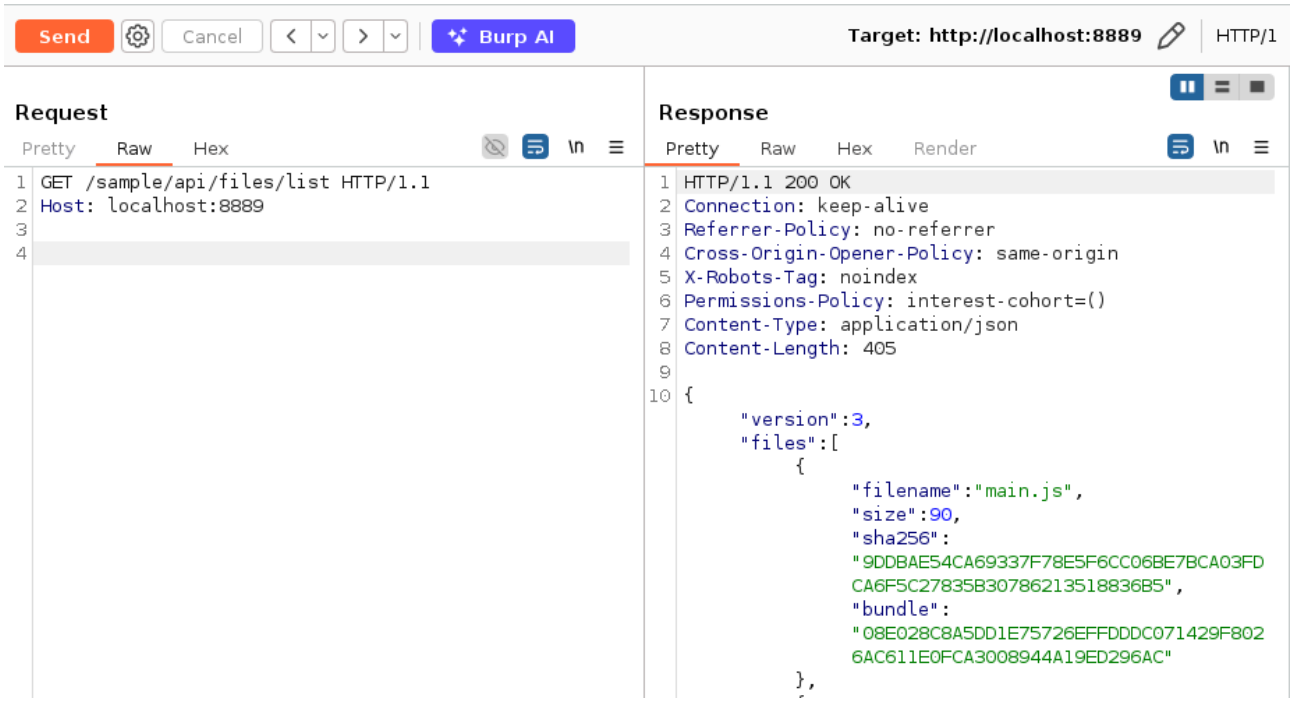


The first request will list all files in the project.

```
GET /sample/api/files/list HTTP/1.1
Host: localhost:8889
```

```
curl --path-as-is -i -s -k -X $'GET' \
    -H $'Host: localhost:8889' \
    $'http://localhost:8889/sample/api/files/list'
```

The response will contain the list of the files:

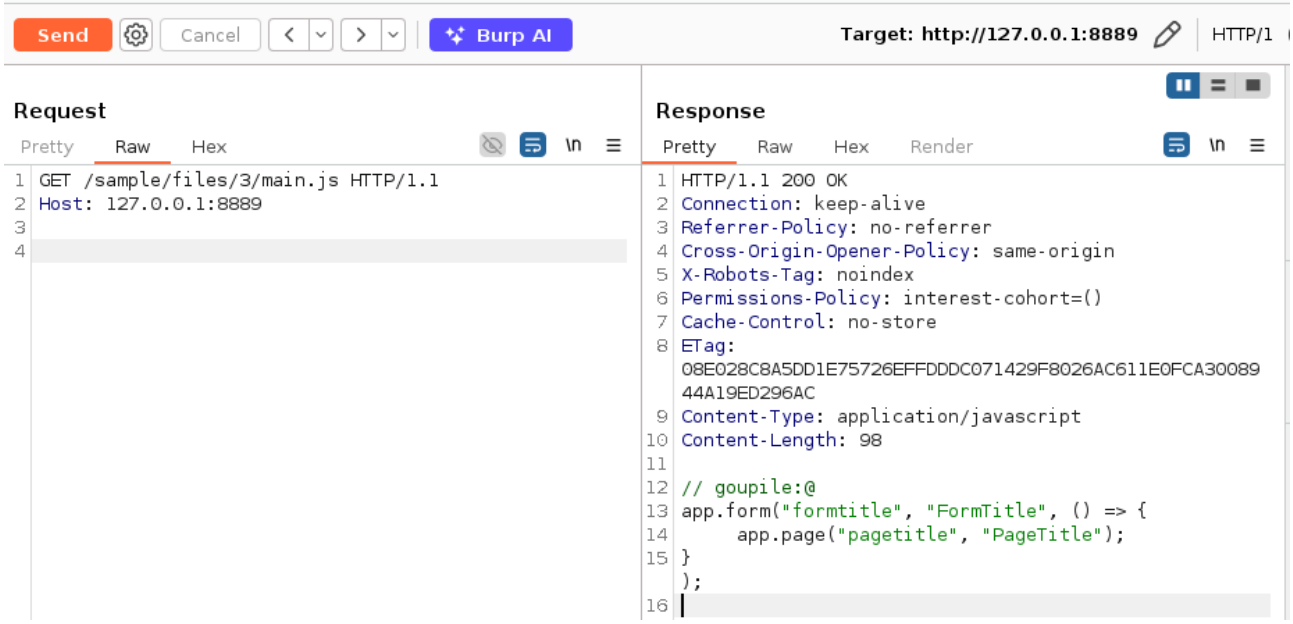


Then we submit a GET request to access the file matching the `version` number received in the first response:

```
GET /sample/files/3/main.js HTTP/1.1
Host: 127.0.0.1:8889
```

```
curl --path-as-is -i -s -k -X $'GET' \
    -H $'Host: 127.0.0.1:8889' \
    $'http://127.0.0.1:8889/sample/files/3/main.js'
```

The response will contain a list of the requested files:



Note from the developer:

> It was made this way on purpose, at least historically (years ago now). At the time, all projects used "offline mode" by default, in which a service worker caches all the project files on startup (before login happens) so projects could work without an internet connection. This is useful for some research projects (such as one which was used for data collection in prison).
> This was not considered a huge problem, because the idea was that the main script, and the page scripts, were not considered more sensitive than the Goupile code itself.
> However, I've revisited this because I saw your issue a few days ago. And things have indeed changed since the original design. Even though the project scripts should not contain sensitive information, in some cases they might. We can play it safe here.
> There's now code to prevent HandleFileList and most HandleFileGet (with a handful of exceptions, such as access to /project/favicon.png, which can be customized per project) calls from succeeding if the user is not assigned to an instance.. unless guests are allowed or offline mode is enabled, in which cases it works like before.

Impact:

A guest user might be able to read the form sources of forms that are not intended to be accessible to guests, and might contain sensitive information.

Recommendation:

- Do not allow guests to access resources that are not set to allow it.
- `HandleFileList` and `HandleFileGet` should check whether the instance allows guest access if there is no current user session.

4.3 CLN-006 — Log Injection

Vulnerability ID: CLN-006

Vulnerability type: CWE-117: Improper Output Neutralization for Logs

Threat level: Low

Description:

The server constructs a log message from user input, but it does not neutralize potentially dangerous elements before writing it to standard output.

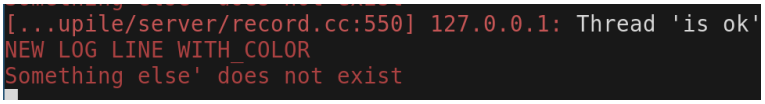
Technical description:

Some APIs log user content directly without any encoding or escaping, like `api/records/get`. To reproduce this issue perform a request similar to this:

```
GET /sample/api/records/get?tid=is+ok'%0A%1b%5b%33%31%6dNEW+LOG+LINE+WITH_COLOR%0aSomething+else
 HTTP/1.1
Host: 127.0.0.1:8889
```

```
curl --path-as-is -i -s -k -X $'GET' \
    -H $'Host: 127.0.0.1:8889' \
    $'http://127.0.0.1:8889/sample/api/records/get?tid=is+ok\'%0A%1b%5b%33%31%6dNEW+LOG+LINE
+WITH_COLOR%0aSomething+else'
```

which creates a log entry like this:



Impact:

An attacker might inject log entries that could hide other attacks or make it more difficult to triage one.

Recommendation:

- Escape, encode, or remove special characters like new lines in user input concatenated with logs.

4.4 CLN-011 — SendTwilio URL encoding

Vulnerability ID: CLN-011

Vulnerability type: CWE-116: Improper Encoding or Escaping of Output

Threat level: Low

Description:

SendTwilio does not use `FmtUrlSafe` to encode the `to` field.

Technical description:

The `to` field comes from the same JSON body input as the `message` field that is escaped. See file `src/goupile/server/message.cc` at line 165.

```
const char *message = nullptr;
    {
        bool success = http_ParseJson(io, Kibibytes(1), [&](json_Parser *json) {
            bool valid = true;

            for (json->ParseObject(); json->InObject(); ) {
                Span<const char> key = json->ParseKey();

                if (key == "to") {
                    json->ParseString(&to);
                } else if (key == "message") {
                    json->ParseString(&message);
                } else {
                    json->UnexpectedKey(key);
                    valid = false;
                }
            }
```

The `SendTwilio` function, in file `src/core/request/sms.cc` at line 79, encodes one field but not the other.

```
bool sms_Sender::SendTwilio(const char *to, const char *message)
{
    BlockAllocator temp_alloc;

    CURL *curl = curl_Init();
    if (!curl)
        return false;
    K_DEFER { curl_easy_cleanup(curl); };

    const char *url = Fmt(&temp_alloc, "https://api.twilio.com/2010-04-01/Accounts/%1/Messages",
 config.authid).ptr;
    const char *body = Fmt(&temp_alloc, "To=%1&From=%2&Body=%3", to, config.from,
 FmtUrlSafe(message, "-._~")).ptr;
```

Impact:

A user that can send messages could alter the request by injecting URL query separators and for example change the `from` value that would normally be read from the config file.

Recommendation:

- Encode the `to` field using `FmtUrlSafe`.

4.5 CLN-012 — Missing HTTP security headers

Vulnerability ID: CLN-012

Vulnerability type: CWE-358: Improperly Implemented Security Check for Standard

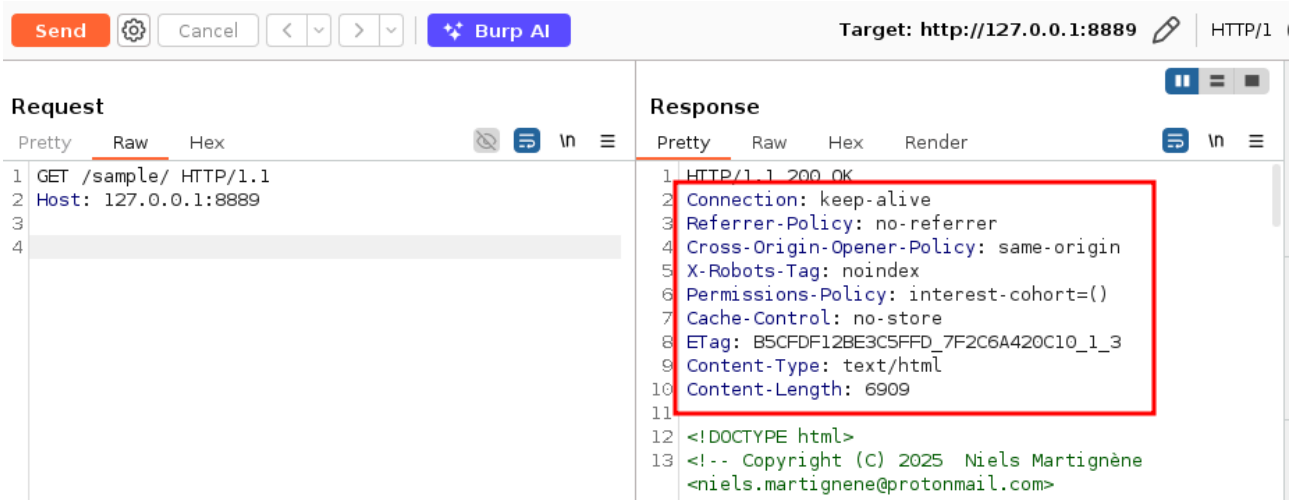
Threat level: Low

Description:

The application does not set three common security headers on the root path, these headers mitigate against common attacks on web application

Technical description:

The HTTP headers set during our test were the following for the root page and for the form pages:



To reproduce this step run this command and observe that the server does not set headers `Strict-Transport-Security`, `Content-Security-Policy`, `X-Content-Type-Options` and `X-Frame-Options`.

```
curl -X GET -I https://target
```

Note that since the server operates in HTTP-only mode, so without encryption it is fine that header `Strict-Transport-Security` is not set.

Another resource to test for header security is the Mozilla HTTP Observatory, where you can scan a URI and see a list of recommendations similar to ours.

Impact:

The lack of a `Strict-Transport-Security` header increases the risk of man-in-the-middle attacks by allowing connections over insecure HTTP. A missing `Content-Security-Policy` (CSP) header exposes the website to cross-site scripting (XSS) and data injection attacks. No `X-Content-Type-Options` permits MIME-sniffing attacks, where the web browser interprets server responses incorrectly. No `X-Frame-Options` allows attackers to frame the website, enabling clickjacking attacks. Similar defences can also be implemented using CSP's `frame-ancestors` property.

Recommendation:

Set the following headers:

1. `Content-Security-Policy` Implement a content security policy. For guidance see https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP.
2. `X-Content-Type-Options` Set to `nosniff`
3. `X-Frame-Options` Set to `SAMEORIGIN` or `DENY`. For guidance see https://developer.mozilla.org/en-US/docs/Web/Security/Practical_implementation_guides/Clickjacking, or implement `frame-ancestors` in CSP.

Setting a `Content-Security-Policy` is the most important step, as the most important security guarantee is to prevent the execution of inline JavaScript, but this also requires the most changes to the application code. For example, the main HTML Goupile page would need to transfer all the inline JavaScript code into a separate script that is loaded into the HTML page.

5 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

5.1 NF-001 — Arbitrary JavaScript in forms is expected

By design, Goupile allows developers to use JavaScript in forms, since a form is created by writing a script.

Only authorized users can write forms, but they can write arbitrary JavaScript code.

For example, it is sufficient to write any JavaScript code in the form editor:

```
console.log('logging from form editor')
form.output(html`<img src=x onerror=console.log('logging from form image')>`)
```

This was not considered an issue after talking with the developers since it is a feature of the framework.

5.2 NF-005 — JSON parser fuzz testing

Fuzzing the JSON parser is an important target since most of the requests have a JSON body and would be a likely entry point for an attacker.

To instrument it with AFL we created a new entry in `FelixBuild.ini` pointing to this file, and built it with the following command, while AFL was installed in on our system:

```
./felix -s -pFast --host=:afl-gcc-fast
```

To start the fuzzing a simple command like the following can be used, where there is at least one JSON file present in the `in` folder.

```
afl -i ./in -o ./out -- ./bin/Fast/json_parser
```

Below we have provided the AFL target used to fuzz the JSON parser. This target was inspired by the code parsing the records submitted by potentially anonymous users.

```
#include "src/core/base/base.hh"
#include "src/core/wrap/json.hh"

namespace K {

// Redefining structs that I was
// not able to import
struct DataConstraint {
    const char *key = nullptr;
    bool exists = false;
    bool unique = false;
};
```

```
struct FragmentInfo {
    int64_t fs = -1;
    const char *eid = nullptr;
    const char *store = nullptr;
    int64_t anchor = -1;
    const char *summary = nullptr;
    bool has_data = false;
    Span<const char> data = {};
    HeapArray<const char *> tags;
};

struct CounterInfo {
    const char *key = nullptr;
    int max = 0;
    bool randomize = false;
    bool secret = false;
};

struct SignupInfo {
    bool enable = false;

    const char *url = nullptr;
    const char *to = nullptr;
    const char *subject = nullptr;
    Span<const char> html = nullptr;
    Span<const char> text = nullptr;
};

struct BlobInfo {
    const char *sha256 = nullptr;
    const char *name = nullptr;
};

// Defining target function that simulate
// the record save json parser
// since this function had many fields and
// could be submitted by anonymous users
void FuzzHandleRecordSave(unsigned char *buf, int len) {
  const char *tid = nullptr;
  FragmentInfo fragment = {};
  HeapArray<DataConstraint> constraints;
  HeapArray<CounterInfo> counters;
  HeapArray<const char *> publics;
  SignupInfo signup = {};
  HeapArray<BlobInfo> blobs;
  bool claim = true;
  bool valid = true;

  BlockAllocator str_alloc;
  StreamReader reader(MakeSpan(buf, len), "<json>");
  json_Parser json(&reader, &str_alloc);

  for (json.ParseObject(); json.InObject();) {
    Span<const char> key = json.ParseKey();

    if (key == "tid") {
      json.ParseString(&tid);
    } else if (key == "eid") {
      json.ParseString(&fragment.eid);
    } else if (key == "store") {
      json.ParseString(&fragment.store);
```

```
    } else if (key == "anchor") {
      json.ParseInt(&fragment.anchor);
    } else if (key == "fs") {
      json.ParseInt(&fragment.fs);
    } else if (key == "summary") {
      json.SkipNull() || json.ParseString(&fragment.summary);
    } else if (key == "data") {
      switch (json.PeekToken()) {
      case json_TokenType::Null: {
        json.ParseNull();

        fragment.data = {};
        fragment.has_data = true;
      } break;
      case json_TokenType::StartObject: {
        json.PassThrough(&fragment.data);
        fragment.has_data = true;
      } break;

      default: {
        LogError("Unexpected value type for fragment data");
        valid = false;
      } break;
      }
    } else if (key == "tags") {
      for (json.ParseArray(); json.InArray();) {
        const char *tag = json.ParseString().ptr;
        fragment.tags.Append(tag);
      }
    } else if (key == "constraints") {
      for (json.ParseObject(); json.InObject();) {
        DataConstraint constraint = {};

        json.ParseKey(&constraint.key);
        for (json.ParseObject(); json.InObject();) {
          Span<const char> type = json.ParseKey();

          if (type == "exists") {
            json.ParseBool(&constraint.exists);
          } else if (type == "unique") {
            json.ParseBool(&constraint.unique);
          } else {
            json.UnexpectedKey(key);
            valid = false;
          }
        }

        constraints.Append(constraint);
      }
    } else if (key == "counters") {
      for (json.ParseObject(); json.InObject();) {
        CounterInfo counter = {};

        json.ParseKey(&counter.key);
        for (json.ParseObject(); json.InObject();) {
          Span<const char> key = json.ParseKey();

          if (key == "key") {
            json.ParseString(&counter.key);
          } else if (key == "max") {
            json.SkipNull() || json.ParseInt(&counter.max);
```

```
        } else if (key == "randomize") {
          json.ParseBool(&counter.randomize);
        } else if (key == "secret") {
          json.ParseBool(&counter.secret);
        } else {
          json.UnexpectedKey(key);
          valid = false;
        }
      }

      counters.Append(counter);
    }
  } else if (key == "publics") {
    for (json.ParseArray(); json.InArray();) {
      const char *key = json.ParseString().ptr;
      publics.Append(key);
    }
  } else if (key == "signup") {
    switch (json.PeekToken()) {
    case json_TokenType::Null: {
      json.ParseNull();
      signup.enable = false;
    } break;
    case json_TokenType::StartObject: {
      signup.enable = true;

      for (json.ParseObject(); json.InObject();) {
        Span<const char> key = json.ParseKey();

        if (key == "url") {
          json.ParseString(&signup.url);
        } else if (key == "to") {
          json.ParseString(&signup.to);
        } else if (key == "subject") {
          json.ParseString(&signup.subject);
        } else if (key == "html") {
          json.ParseString(&signup.html);
        } else if (key == "text") {
          json.ParseString(&signup.text);
        } else {
          json.UnexpectedKey(key);
          valid = false;
        }
      }
    } break;

    default: {
      LogError("Unexpected value type for signup data");
      valid = false;
    } break;
    }
  } else if (key == "blobs") {
    for (json.ParseArray(); json.InArray();) {
      BlobInfo blob = {};

      for (json.ParseObject(); json.InObject();) {
        Span<const char> key = json.ParseKey();

        if (key == "sha256") {
          json.ParseString(&blob.sha256);
        } else if (key == "name") {
```

```
            json.ParseString(&blob.name);
          } else {
            json.UnexpectedKey(key);
            valid = false;
          }
        }

        blobs.Append(blob);
      }
    } else if (key == "claim") {
      json.ParseBool(&claim);
    } else {
      json.UnexpectedKey(key);
      valid = false;
    }
  }
}
} // namespace K

__AFL_FUZZ_INIT();

int main(int argc, char **argv) {
  unsigned char *buf = __AFL_FUZZ_TESTCASE_BUF;
  while (__AFL_LOOP(10000)) {
    int len = __AFL_FUZZ_TESTCASE_LEN;
    K::FuzzHandleRecordSave(buf, len);
  }
  return 0;
}
```

5.3 NF-008 — HTTP server fuzzing setup

During the testing we created a profile for fuzzing the HTTP request target, using AFLplusplus.

It was too complicated to separate the HTTP parsing logic from the web server and database one, so the target uses the full Goupile code, which slows down the fuzzing.

Another difficulty was trying to make the target stable; parsing more than one HTTP request made it very unstable, often reaching 0%, which means we are not doing any coverage-guided fuzzing.

This setup heavily modifies the main Goupile server removing all the unnecessary code and by piping the standard input to a socket read by the server.

The following snippet is part of the modified `RunServe` function calling function `write_stdin` that reads from the AFL buffer into the socket:

```
static int RunServe(K::Span<const char *> arguments) {
  K::BlockAllocator temp_alloc;

  const char *config_filename = "goupile.ini";

  {
    K::OptionParser opt(arguments, K::OptionMode::Skip);

    while (opt.Next()) {
      if (opt.Test("-C", "--config_file", K::OptionType::Value)) {
```

```
      if (K::IsDirectory(opt.current_value)) {
        config_filename =
            Fmt(&temp_alloc, "%1%/goupile.ini",
                K::TrimStrRight(opt.current_value, K_PATH_SEPARATORS))
                .ptr;
      } else {
        config_filename = opt.current_value;
      }
    } else if (opt.TestHasFailed()) {
      return 1;
    }
  }
}

if (!K::LoadConfig(config_filename, &K::gp_config))
  return 1;

const char * sock_name;
// Parse arguments by adding a socket option so that
// each AFL instance can fuzz using a different one
{
  K::OptionParser opt(arguments);

  while (opt.Next()) {
    if (opt.Test("-C", "--config_file", K::OptionType::Value)) {
      // Already handled
    } else if (opt.Test("-p", "--port", K::OptionType::Value)) {
      if (!K::gp_config.http.SetPortOrPath(opt.current_value))
        return 1;
    } else if (opt.Test("-s", "--sock", K::OptionType::Value)) {
      sock_name = opt.current_value;
      K::gp_config.http.SetProperty("UnixPath", sock_name, "/");
    } else if (opt.Test("--sandbox")) {
      // sandbox = false;
    } else {
      opt.LogUnknownError();
      return 1;
    }
  }

  opt.LogUnusedArguments();

  if (!K::gp_config.Validate())
    return 1;
}

K::InitAssets();

// Make sure tmp and instances live on the same volume, because we need to
// perform atomic renames in some cases.
{
  const char *tmp_filename1 =
      K::CreateUniqueFile(K::gp_config.tmp_directory, nullptr, ".tmp", &temp_alloc);
  if (!tmp_filename1)
    return 1;
  K_DEFER { K::UnlinkFile(tmp_filename1); };

  const char *tmp_filename2 = K::CreateUniqueFile(K::gp_config.instances_directory,
                                                  "", ".tmp", &temp_alloc);
  if (!tmp_filename2)
    return 1;
```

```
    K_DEFER { K::UnlinkFile(tmp_filename2); };

    if (K::RenameFile(tmp_filename1, tmp_filename2, (int)K::RenameFlag::Overwrite) !=
        K::RenameResult::Success)
      return 1;
  }

  if (!K::OpenMainDatabase())
    return 1;

#if defined(__linux__)
  if (!K::NotifySystemd())
    return 1;
#endif

  K::http_Daemon daemon;

  unsigned char *buf = __AFL_FUZZ_TESTCASE_BUF;
  while (__AFL_LOOP(10000)) {
    int afl_len = __AFL_FUZZ_TESTCASE_LEN;


   if (!daemon.Bind(K::gp_config.http))
    return 1;

    if (!K::InitDomain())
      return 1;

    if (!daemon.Start(K::HandleRequest))
      return 1;

    write_stdin(sock_name, buf, afl_len);

    daemon.Stop();

  }

  K::CloseDomain();

  return 0;
}
```

6 Future Work

- **Perform fuzz testing targeting the HTTP server request parser**
  Ensuring that the HTTP server can parse even malformed HTTP requests without memory issues will improve the server's security.
- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, perform a repeat test to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is a process that must be continuously evaluated and improved; this penetration test is just a single snapshot. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security.

7 Conclusion

We discovered 1 Elevated, 1 Moderate and 3 Low-severity issues during this penetration test.

The audit found only two security issues that could have direct impact on the system. The low-level issues might provide an attacker with a foothold or cover for more advanced post-exploitation activity.

Apart from the identified flaws, we judged the majority of the code to be sound. The audit highlighted the importance of integrating external libraries for cryptography, database access, and SMS handling.

The report also documents the creation of a rudimentary fuzz-testing environment targeting the JSON parser and HTTP server. Although we found only one issue with fuzzing, it remains a valuable technique for uncovering edge-case memory-management errors in a non-memory-safe language like C++.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a process that must be continuously evaluated and improved – this penetration test is just a one-time snapshot. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing Team

Andrea Jegher	Andrea is a security engineer with experience in offensive security and secure development. He started his career focusing on Web Application as a developer and as a penetration tester. Later he studied other fields of security such as cloud, networks and desktop applications.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.