Updates and Responses to Stegotorus Pen Test Report by Radically Open Security

Stegotorus Project

March 15, 2020

Contents

1 Introduction

After receiving the code review, we studied all the finding from the ROS report. We fixed most of the issues including those which were concerning dead codes (not running in Stegotorus). Those issues that have been fixed have a corresponding commit on github. The following is the detailed response to the audit findings:

2 Fixed issues in the actual used code

1. 001 — Potential Remote Code Execution in `src/network.cc`

2. 006 — Stack-Based Buffer Overflow in `PayloadScraper::scrape_url()`

3. 007 — Off by One in `modus_operandi_t::process_command_line_config()`

4. 010 — Potential Off by One in `PayloadServer::find_uri_type()`

5. 013 — Duplicated Code in the `JSSteg` Class (duplicated code deleted)

6. 014 — Signal-Unsafe Function Used in `lethal_signal()`

7. 016 — Undetected Compression Errors in `JSSteg::encode()`

8. 017 — Integer Underflow in `JPGSteg::starting_point()`

   This was fixed but would only pertain to cover file over 2GB size which Stegotorus never transfers.

9. 018 — Temporary Filename Issue in `PayloadScraper::scrape()`

3 Fixed issues in the dead code never run in Stegotorus and therefore we did not review in advance:

1. 002 — Dangerous behaviour in `gen_uri_field()`

2. 005 — Dangerous Behaviour in `mkem.cc` (dead code removed)

3. 009 — Potential Crash in `embed_steg_t::receive()`

4. 011 — Potential Invalid Access and Information Leak in `http_steg_t::http_client_uri_transm`

4 Issues considered not to be valid

4.1 003 — Insecure File Transfers From Remote Apache Hosts in `PayloadScraper::scrape()` and `PayloadScraper::compute_capacity()`

We do not believe that this is a vulnerability as the payload scraper is contacting the cover server. It does not matter who is the cover server and what content it serves us. We may as well scrape the cover content from arbitrary observed Internet traffic (as http steg mod does). We do not rely on the cover content. If the content does not correspond to the type we have

requested, the corresponding file steg module fails to encode content in it and will be flagged as unusable. As such, we do not trust the cover server in any way and so do not need to authenticate the authenticity of its content. Q.E.D.

4.2 004 — Data Corruption in `curl_read_data_cb()`

"First, the result of the multiplication of the two size_t parameters size and nmemb would potentially result in an integer overflow, when the result is stored in a variable of the same size (here $\text{no}_{\text{bytes2read}}$). Fortunately, the CURL API documents that size will always be 1. However, this may change in a future release and the code should be updated to handle this situation safely."

We do not think that this happens because curl expects to return the actual number of bytes read in size_t therefore it does not make sense for curl to make more bytes available than actually we can read, that would be considered a bug in curl.

in case, curl will change the prototype and Stegotorus will not compile as is.

"Then, another mistake was apparently made, where $\text{no}_{\text{bytes2read}}$ is multiplied again by size when populating the underlying stringstream. Luckily, again, size should be 1, therefore avoiding a bad consequence."

fixed.

"Regardless, the data provided by CURL comes from a remote server, and may contain NUL characters (0x00). In this case, the incoming data will probably be cut short at the first occurrence of this character."

This does not occur according to the c++ ref doc and we tested it and it doesn't happen:

```
#include <iostream>
#include <string>
#include <sstream>
#include <memory.h>

using namespace std;

int main()
{
  char* teststring[100 ];
```

```
  stringstream sStream;
  memcpy(teststring, "test1\0test2\0test3\0", 15);
  sStream.write((const char*)teststring, 15);

  cout << sStream.str().length() << endl;
}
/// from C++ ref doc of stringstream.write:
// This function simply copies a block of data, without checking its contents:
//The array may contain null characters, which are also copied without stopping the co

//output
//g++ streamwrite.cpp
//[user@machine test]$ ./a.out
//15
```

"If the incoming data is long enough, this callback will be called again, with the same effect, corrupting the result in memory some more.

Since this callback returns `no_bytes_2_read` without obtaining errors from the stringstream object, CURL cannot detect this issue in the callback, and will keep proceeding as if no error was made."

It does:

```
if( ((stringstream*)userp)->bad()){
  log_debug("Error reading data from curl");
  return 0;
}
```

4.3 008 — Undefined Behaviour From curl in `ApachePayloadServer::ApachePayloadS`

"The constructor for the `ApachePayloadServer` class will only initialize the curl handler if it is set to NULL at that time."

This is not what this line:

```
if (!(_curl_obj = curl_easy_init()))
  log_abort("Failed to initiate the curl object");
```

does. It tries to initialize the `_curl_obj` no matter what and if it fails it aborts the execution of Stegotorus. So this:

"From the lines 116 on, if `_curl_obj` was not actually initialized as intended, the behaviour of the program will be undefined."

does not happen. [Q.E.D]

4.4 012 — Missing Input Validation in `find_content_length()`

This has been resolved before the audit, auditor seems to have been looking at an older version of the code.

4.5 015 — Potential Crash in `chop_config_t::conn_create()`

repetition of issue 1