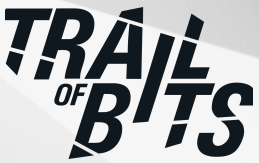

cURL

Security Assessment

December 20, 2022

Prepared for:
Daniel Stenberg, cURL
Open Source Security Foundation (OpenSSF)
Open Source Technology Improvement Fund

Prepared by: **Emilio López, Spencer Michaels, and Paweł Płatek**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

The Linux Foundation, via OpenSSF and strategic partner Open Source Technology Improvement Fund, engaged Trail of Bits to review the security of cURL. From September 12 to October 7, 2022, a team of four Trail of Bits consultants conducted a security review of the client-provided source code, with five and a half engineer-weeks of effort. Since this project coincided with a Trail of Bits Maker Week, six additional people contributed five additional days of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the system. We had access to the cURL source code, documentation, and fuzzing harnesses. We performed static and dynamic automated and manual testing of the target system and its codebase, using both automated and manual processes.

Summary of Findings

The audit uncovered a small number of significant flaws that could impact system confidentiality, integrity, or availability. A summary of the findings and details on notable findings are provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	2
Low	5
Informational	6
Undetermined	2

CATEGORY BREAKDOWN

Category	*Count*
Configuration	4
Cryptography	1
Data Validation	8
Denial of Service	1
Undefined Behavior	1

Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

- **TOB-CURL-7: Double free when using HTTP proxy with specific protocols**
  A double free occurs during connection cleanup when cURL (or libcurl) makes a connection through a proxy.
- **TOB-CURL-10: Use after free while using parallel option and sequences**
  Using cURL in parallel mode (-Z) with two consecutive sequences followed by an empty bracket results in a use after free.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

Derek Zimmer, Program Manager
derek@ostif.org

Amir Montazery, Program Manager
amir@ostif.org

The following engineers were associated with this project:

Spencer Michaels, Consultant
spencer.michaels@trailofbits.com

Emilio López, Consultant
emilio.lopez@trailofbits.com

Anders Helsing, Consultant
anders.helsing@trailofbits.com

Paweł Płatek, Consultant
pawel.platek@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 2, 2022	Pre-project kickoff call
September 20, 2022	Status update meeting #1
September 27, 2022	Status update meeting #2
October 4, 2022	Status update meeting #3
October 11, 2022	Delivery of report draft; report readout meeting
December 13, 2022	Delivery of final report with fix review
December 20, 2022	Re-delivery of final report with fix review

Project Goals

The engagement was scoped to provide a security assessment of cURL. Specifically, we sought to answer the following non-exhaustive list of questions:

- Does cURL safely handle inputs from untrusted sources?
- Does cURL properly implement the security-relevant features of each protocol it supports, both in terms of implementation details and API compliance with the protocol specification?
- Can an attacker exploit cURL's connection reuse mechanism to hijack existing connections created by other users in applications that use libcurl?
- Does cURL safely store and transmit credentials and other secrets passed to it?
- Do cURL fuzz tests provide coverage in likely areas of issues, such as parsing or decoding, or areas where vulnerabilities have been found previously?
- Do cURL fuzzers run against different supported build configurations?
- Do cURL fuzzers leverage techniques and tools to make fuzzing more effective, such as dictionaries and structure-aware fuzzing?

Project Targets

The engagement involved a review and testing of the targets listed below.

cURL

Repository	https://github.com/curl/curl
Version	`2ca0530a4d4bd1e1ccb9c876e954d8dc9a87da4a`
Type	Library and CLI binary
Platform	Native

cURL fuzzer for OSS-Fuzz

Repository	https://github.com/curl/curl-fuzzer
Version	`32fc19a4fa6398ff9a23b744c7cf21547b375c6a`
Type	Fuzzing harnesses and scripts
Platform	x86 and x86_64

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches and include the following:

- Automated analysis of the cURL codebase with static analyzers such as Semgrep and CodeQL, with manual follow-up to investigate promising leads.
- Targeted manual code review of the following components:
    - X.509 certificate parser
    - Cookie storage and cookie file parsing
    - URI parsing
    - HSTS data storage
    - cURL's custom printf implementation (`mprintf`)
    - Connection reuse
    - HTTP redirects
    - Authentication header handlers
    - `Alt-Svc` header handlers
    - MQTT protocol implementation
    - The cURL command-line tool
- Fuzzing of the built-in X.509 certificate parser with AFL
- Review of the existing fuzzing harnesses and coverage information, and identification of areas for improvement
- Improvement of existing harnesses to enhance coverage and test a larger set of cURL functionalities
- Development of new targeted harnesses targeting code paths which may be sources of issues, such as parsers and decoders
- Fuzzing of a cURL binary built with the ASan address sanitizer option enabled

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- We partially reviewed shared logic and some protocol implementations, but due to time constraints, we were unable to exhaustively review cURL's supported protocols, nor did we have time to fully evaluate the standards-compliance of the protocol implementations, except in a few specific known cases. It may be beneficial to conduct a targeted review against individual protocols of special concern.
- We did not review the correctness of integrations with various SSL libraries, i.e., the vTLS component.
- We evaluated fuzzing coverage and functionality using a specific, common build configuration of cURL that included OpenSSL 1.0.2m and Nghttp2 1.41.0 on x86_64 Linux. cURL can be built with support for a wide variety of libraries and target systems, and fuzzing with different configuration sets may be beneficial.

Fuzzing Coverage Assessment

As part of this engagement, Trail of Bits reviewed the cURL project's fuzz tests and their coverage, with the aim of improving their depth and coverage. The libcurl library is continuously fuzzed by OSS-Fuzz, an initiative for fuzzing open-source software, using scripts and harnesses from the `curl-fuzzer` repository.

Approach

As a first step, we reviewed the coverage currently achieved by the fuzzing harnesses in the repository, combined with the seed cases. We also reviewed a coverage report from OSS-Fuzz. These reports showed several areas in cURL that lacked fuzzing coverage, such as Alt-Svc and HSTS management, and DNS-over-HTTPS, among others.

We then proceeded to make changes to the `curl-fuzzer` repository to improve coverage of said areas and try to detect new bugs. This resulted in the discovery of TOB-CURL-3 and TOB-CURL-7. A summary of the harness improvements and new harnesses can be found below. We recommend including these modifications as part of the OSS-Fuzz cURL harness suite.

Fuzzing harness changes	
Harness	**Description**
curl_fuzzer	Addition of Alt-Svc and HSTS header testing, and multiple CURLOPTs
curl_fuzzer_altsvc	Fuzzing Alt-Svc header parsing
curl_fuzzer_base64	Fuzzing base64 encoding and decoding
curl_fuzzer_doh	Fuzzing DNS-over-HTTPS (DoH) decoding
curl_fuzzer_escape	Fuzzing escape/unescape functions
curl_fuzzer_parsedate	Fuzzing date parsing

Alt-Svc header parsing

Rationale

HTTP servers may send `Alt-Svc` headers to indicate support for alternative protocols, such as HTTP/2. This facilitates connection protocol upgrades. cURL supports parsing these headers and storing the information collected in a file for future reference.

The current fuzzing coverage did not show any coverage for the relevant code (`altsvc.c`). This code also does manual string parsing, which tends to be error-prone and is hard for a human reviewer to reason about, so it is a good target for fuzzing.

Harness

We implemented a simple harness that receives a string representing an `Alt-Svc` header and parses it with `Curl_altsvc_parse`. We seeded the fuzzer with some of the `Alt-Svc` strings from unit test #1654. This improved the coverage of the file to over 50% of lines and functions. We executed this harness for over a week with address sanitizer (ASan) enabled, but it did not find any failures.

Future work

`altsvc.c` also deals with storing, loading, and looking up `Alt-Svc` entries from a Curl-generated file store. These functions are also currently uncovered. We recommend enhancing the harness suite to exercise these functions as well.

Base64 encoding and decoding

Rationale

Base64 decoding and encoding are widely used across cURL, including in several authentication schemas, certificate management, WebSockets, HTTP connection upgrades, and LDAP support.

Although the current fuzzing coverage of the code in `base64.c` code is high, there was no explicit harness fuzzing the encoder and decoder in a roundtrip. Encoding and decoding a series of bytes should always result in the same message; fuzzing in roundtrip is a good way to ensure that the implementations do not corrupt the data.

Harness

We implemented a harness that receives a string and tries to decode it as Base64. If decoding succeeds, it will re-encode and decode the buffer and ensure that the data stays unchanged. We executed this harness for over a week with address sanitizer (ASan) enabled, but it did not find any failures.

DoH decoding

Rationale

cURL supports resolving domain names in URLs using DNS-over-HTTPS (DoH) instead of traditional DNS requests. The current fuzzing coverage did not show any coverage for the relevant code (`doh.c`). This code also does manual parsing of external inputs, so it is a good target for fuzzing.

Harness

We implemented a harness that receives a buffer and tries to decode it as a DoH response using `doh_decode`. We executed this harness for over a week with address sanitizer (ASan) enabled, but it did not find any failures.

Future work

doh.c also deals with encoding requests, and these functions are also currently uncovered. We recommend enhancing the harness suite to exercise these functions as well.

Escape

Rationale

Creating a round-trip fuzzing harness is a good practice, as it may detect a problem when the implementation changes and helps to ensure that the current implementation is correct. The cURL URL escaping and unescaping functions (`curl_easy_escape` and `curl_easy_unescape` respectively) were not fuzz tested in this way.

Harness

We implemented a simple harness, escaping input (using `curl_easy_escape`) and unescaping the result (with `curl_easy_unescape`) to confirm that those functions combined keep the input data intact.

Date parsing

Rationale

Date string parsing is also widely used across cURL, including in the HTTP implementation (used for HSTS, Alt-Svc, and cookies) and other protocols such as FTP and SSH.

Although the current fuzzing coverage of the code in `parsedate.c` code is high, there was no explicit harness fuzzing this functionality. As the function does complex string parsing and handles user input, we felt it would be a good target for a specific harness.

Harness

We implemented a harness that receives a string and tries to parse it using `Curl_getdate_capped`. We executed this harness for over a week with address sanitizer (ASan) enabled, but it did not find any failures.

Expanding HSTS and Alt-Svc coverage

As cURL, by default, accepts only the `Strict-Transport-Security` and `Alt-Svc` headers when sent over TLS, the OSS-Fuzz HTTP fuzzer was missing most code paths in `altsvc.c` and `hsts.c`. To expand coverage of these files, we made the following changes:

- After noting that the `CURL_ALTSVC_HTTP` environment variable informs debug builds of cURL to accept the `Alt-Svc` header over unencrypted HTTP, we enabled the variable in the `curl-fuzzer` codebase.
- We added an analogous variable for HSTS (`CURL_HSTS_HTTP`) to the cURL codebase and enabled it in the `curl-fuzzer` codebase (see Appendix D).
- We modified `curl_fuzzer.cc` to set `CURLOPT_HSTS` and `CURLOPT_ALTSVC` to point to their respective cache files (in this case, both `/dev/null`).

These changes enabled us to significantly expand the OSS-Fuzz coverage for `altsvc.c` and `hsts.c`. Because of the lack of stateful fuzzing, many of the areas still without coverage are related to the loading and processing of the HSTS and `Alt-Svc` cache files from disk. We recommend adding direct fuzz harnesses for such functions (e.g., `hsts_add()`, `hsts_push()`, `hsts_pull()`).

Expanding file-related and authentication-related fuzzing coverage

We added TLVs and test cases to several of the existing `curl-fuzzer` harnesses where `codecoverage.sh` showed pre-existing tests exercised < 50% of a file. We focused on improving areas like HTTP authentication methods and domain-specific file and filename parsing (e.g., cache files for the `Strict-Transport-Security` and `Alt-Svc` headers).

We added corpus seed cases that covered the following:

- Additional WebSockets functionality, including upgrade from HTTP;
- Areas that team members noted were performing unexpectedly;
- Previously uncovered functionality involved in recent CVEs, including but not limited to CVE-2022-22576, CVE-2022-35252, and CVE-2022-32207.

To make it easier to add test coverage in these areas, we also did the following:

- Added multiple TLV types relating to `CURLOPT_` options to the test case generation scripts to enable fuzzing differently authenticated messages across various protocols.
- Added basic fuzzing directly to `curl_fuzzer.cc` and `curl_fuzzer_tlv.cc` for `CURLOPT_` options, which take files such as `.netrc` and the read-only cookie file specified with `CURLOPT_COOKIEFILE` when we added coverage for the HSTS and Alt-Svc cache file paths.
- After noting it was moderately difficult to maintain a mental map matching string authentication method names `#defined` in `curl.h` e.g. `CURLAUTH_ONLY` and their unsigned long values (e.g., `` `((unsigned long)1) << 31` ``), we also added an enum class to translate from authentication method names (read in to `generate_corpus.py`) and the unsigned long authentication method values that libcurl expects.

These changes increase general coverage for cURL, but we believe it is preferable to have at least one corpus test case to exercise each `CURLOPT_` option, and ideally also to cover more common combinations of options.

Strategic fuzzing recommendations

We recommend the following general changes to improve the coverage and efficiency of cURL's fuzzing setup:

- **Add dictionaries for other protocols to libFuzzer and OSS-Fuzz**: Adding a dictionary with common words greatly improves the efficiency of fuzzing in certain cases, such as text-based protocols.
- **Ensure that all build configurations (e.g., non-OpenSSL builds) are covered in the fuzz tests**.
- **Add a round-trip fuzzing harness for every encoder/decoder pair**: This will ensure that the encoding and decoding processes work as expected and that data is not corrupted or otherwise modified.
- **Implement structure-aware fuzzing**: `curl-fuzzer` currently uses a type-length-value (TLV) format for inputs in order to encode various types and components of requests and responses. However, as libFuzzer is not aware of the TLV structure, many of the mutations it generates are invalid at the TLV-unpacking stage and have to be discarded by `curl-fuzzer`. This reduces fuzzing efficiency. In accordance with Google's recommendation above, we recommend implementing structure-aware fuzzing by adding a custom mutator that ensures the fuzzer always receives a valid input. There is an open pull request from 2019 to add such a mutator, but its current status is unclear.
- **Cover `argv` fuzzing**: Fuzzing the curl binary with different options can be useful to discover issues such as TOB-CURL-10. This can be achieved using the `argv-fuzz-inl.h` header from the AFL++ project to build the arguments array from standard input in cURL. Also, consider adding a dictionary with possible options and protocols to the fuzzer based on the source code or cURL's manual.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	Where applicable, arithmetic operations are checked carefully for overflows and other exceptional conditions.	**Satisfactory**
Auditing	cURL can produce detailed, verbose debug output.	**Satisfactory**
Authentication / Access Controls	Although no issues pertaining to authentication were discovered in this audit, cURL has had several recent CVEs that could leak credentials over the network (CVE-2022-27776, CVE-2021-22876, and CVE-2020-8169), indicating a potential weak point in this area.	**Moderate**
Complexity Management	cURL is necessarily an extremely complex project, but its codebase is well organized: protocol-specific functionality is isolated, and common functionality is generic and broadly accessible.	**Strong**
Configuration	Some of cURL's options default to the most *common* case rather than the most *secure* one, for the sake of ease of use. In addition, different command-line flags can behave differently—e.g., repeating the same flag may override previous values in some cases and append to them in others—which may result in user confusion.	**Moderate**
Cryptography and Key Management	Wherever technically feasible, cURL relies on well audited third-party libraries such as OpenSSL to perform cryptographic operations. In rare cases—generally uncommon build configurations which do not permit the	**Strong**

	use of such libraries—cURL sparingly uses its own implementations. Our fuzz testing of portions of the latter code (e.g., the X.509 parser) revealed no issues.	
Data Handling	Although data received over the network is generally handled with the appropriate care (with the notable exception of TOB-CURL-5), command-line options and parameters supplied to libcurl are less well checked (TOB-CURL-2, TOB-CURL-3, TOB-CURL-7, TOB-CURL-10). A few cases were discovered where received data is accepted even if its values make no sense within the protocol specification, such as negative HTTP status codes or empty transfer encoding lists.	**Moderate**
Documentation	Documentation for end-users of both the cURL command-line tool and the libcurl library is extensive. However, the documentation for less common protocols such as MQTT does not appear to have been kept up to date, which may indicate a lack of a systematic process to ensure that documentation and code do not go out-of-sync with each other.	**Moderate**
Maintenance	Consistent standards are applied rigorously throughout the codebase; for instance, the handle to the current connection is named identically in any function that takes such a value as a parameter. A small number of outdated comments were discovered, but none pertain to important functionality.	**Satisfactory**
Memory Safety and Error Handling	Several possible memory violations were discovered during the audit, and though they are not likely to be present in cURL's most common use cases, an attacker could easily discover them by fuzzing the cURL codebase.	**Weak**
Testing and Verification	Both unit and fuzz tests are present, but coverage and efficiency could be improved through the use of extra dictionaries, structure aware fuzzing, targeted harnesses, and a wider set of build configurations.	**Moderate**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Bad recommendation in libcurl cookie documentation	Configuration	Informational
2	Libcurl URI parser accepts invalid characters	Data Validation	Undetermined
3	Libcurl Alt-Svc parser accepts invalid port numbers	Data Validation	Undetermined
4	Non-constant-time comparison of secrets	Cryptography	Low
5	Tab injection in cookie file	Data Validation	Informational
6	Standard output/input/error may not be opened	Data Validation	Informational
7	Double free when using HTTP proxy with specific protocols	Data Validation	High
8	Some flags override previous instances of themselves	Configuration	Informational
9	Cookies are not stripped after redirect	Configuration	Low
10	Use after free while using parallel option and sequences	Data Validation	High
11	Unused memory blocks are not freed resulting in memory leaks	Denial of Service	Low
12	Referer header is generated in insecure manner	Configuration	Low

13	Redirect to localhost and local network is possible (Server-side request forgery like)	Data Validation	**Informational**
14	URL parsing from redirect is incorrect when no path separator is provided	Undefined Behavior	**Low**

Detailed Findings

1. Bad recommendation in libcurl cookie documentation

Severity: **Informational**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-CURL-1
Target: `https://everything.curl.dev/libcurl-http/cookies`	

Description
The libcurl documentation recommends that, to enable the cookie store with a blank cookie database, the calling application should use the `CURLOPT_COOKIEFILE` option with a non-existing file name or plain " ", as shown in figure 1.1. However, the former recommendation—a non-blank filename with a target that does not exist—can have unexpected results if a file by that name is unexpectedly present.

Enable cookie engine with reading

Ask libcurl to import cookies into the easy handle from a given file name with the `CURLOPT_COOKIEFILE` option:

```
curl_easy_setopt(easy, CURLOPT_COOKIEFILE, "cookies.txt");
```

A common trick is to just specify a non-existing file name or plain "" to have it just activate the cookie engine with a blank cookie store to start with.

Figure 1.1: The recommendation in libcurl's documentation

Exploit Scenario
An inexperienced developer uses libcurl in his application, invoking the `CURLOPT_COOKIEFILE` option and hard-coding a filename that he thinks will never exist (e.g., a long random string), but which *could* potentially be created on the filesystem. An attacker reverse-engineers his program to determine the filename and path in question, and then uses a separate local file write vulnerability to inject cookies into the application.

Recommendations
Short term, remove the reference to a non-existing file name; mention only a blank string.

Long term, avoid suggesting "tricks" such as this in documentation when a misuse or misunderstanding of them could result in side effects of which users may be unaware.

2. Libcurl URI parser accepts invalid characters

Severity: **Undetermined**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-CURL-2
Target: URL parser	

Description

According to RFC 3986 section 2.2, “Reserved Characters,”

```
reserved    = gen-delims / sub-delims

gen-delims  = ":" / "/" / "?" / "#" / "[" / "]" / "@"

sub-delims  = "!" / "$" / "&" / "'" / "(" / ")"
            / "*" / "+" / "," / ";" / "="
```

Figure 2.1: Reserved characters for URIs.

Furthermore, the host field of the URI is defined as follows:

```
host        = IP-literal / IPv4address / reg-name

reg-name    = *( unreserved / pct-encoded / sub-delims )
...
unreserved  = ALPHA / DIGIT / "-" / "." / "_" / "~"
sub-delims  = "!" / "$" / "&" / "'" / "(" / ")"
               / "*" / "+" / "," / ";" / "="
```

Figure 2.2: Valid characters for the URI host field

However, cURL does not seem to strictly adhere to this format, as it accepts characters not included in the above. This behavior is present in both libcurl and the cURL binary. For instance, characters from the `gen-delims` set, and those not in the `reg-name` set, are accepted:

```
$ curl -g "http://foo[]bar" # from gen-delims
curl: (6) Could not resolve host: foo[]bar

$ curl -g "http://foo{}bar" # outside of reg-name
curl: (6) Could not resolve host: foo{}bar
```

Figure 2.3: Valid characters for the URI host field

The exploitability and impact of this issue is not yet well understood; this may be deliberate behavior to account for currently unknown edge-cases or legacy support.

Recommendations

Short term, determine whether these characters are being allowed for compatibility reasons. If so, it is likely that nothing can be done; if not, however, make the URI parser stricter, rejecting characters that cannot appear in a valid URI as defined by RFC 3986.

Long term, add fuzz tests for the URI parser that use forbidden or out-of-scope characters.

3. libcurl Alt-Svc parser accepts invalid port numbers

Severity: **Undetermined**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-CURL-3
Target: Alt-Svc parser	

Description

Invalid port numbers in `Alt-Svc` headers, such as negative numbers, may be accepted by libcurl when presented by an HTTP server. libcurl uses the `strtoul` function to parse port numbers in `Alt-Svc` headers. This function will accept and parse negative numbers and represent them as unsigned integers without indicating an error.

For example, when an HTTP server provides an invalid port number of -18446744073709543616, cURL parses the number as 8000:

```
* Using HTTP2, server supports multiplexing
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade: len=0
* Using Stream ID: 1 (easy handle 0x12d013600)
> GET / HTTP/2
> Host: localhost:2443
> user-agent: curl/7.79.1
> accept: */*
>
< HTTP/2 200
< server: basic-h2-server/1.0
< content-length: 130
< content-type: application/json
* Added alt-svc: localhost:8000 over h3
< alt-svc: h3=":-18446744073709543616"
<
```

Figure 3.1: Example cURL session

Exploit Scenario

A server operator wishes to target cURL clients and serve them alternative content. The operator includes a specially-crafted, invalid `Alt-Svc` header on the HTTP server responses, indicating that HTTP/3 is available on port `-18446744073709543616`, an invalid, negative port number. When users connect to the HTTP server using standards-compliant HTTP client software, their clients ignore the invalid header. However, when users connect using cURL, it interprets the negative number as an unsigned integer and uses the resulting port number, `8000`, to upgrade the next connection to HTTP/3. The server operator hosts alternative content on this other port.

Recommendations

Short term, improve parsing and validation of `Alt-Svc` headers so that invalid port values are rejected.

Long term, add fuzz and differential tests to the `Alt-Svc` parsing code to detect non-standard behavior.

4. Non-constant-time comparison of secrets

Severity: **Low**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-CURL-4

Targets:
- `lib/url.c:972-973,1133-1134,1382-1383`
- `lib/vtls/vtls.c:149-150`
- `lib/vauth/digest_sspi.c:434-435`
- `lib/netrc.c:219`

Description
Several cases were discovered in which possibly user-supplied values are checked against a known secret using non-constant-time comparison. In cases where an attacker can accurately time how long it takes for the application to fail validation of submitted data that he controls, such behavior could leak information about the secret itself, allowing the attacker to brute-force it in linear time.

In the example below, credentials are checked via `Curl_safecmp()`, which is a memory-safe, but not constant-time, wrapper around `strcmp()`. This is used to determine whether or not to reuse an existing TLS connection.

```
#ifdef USE_TLS_SRP
     Curl_safecmp(data->username, needle->username) &&
     Curl_safecmp(data->password, needle->password) &&
     (data->authtype == needle->authtype) &&
#endif
```

Figure 4.1: `lib/url.c`, lines 148 through 152. Credentials checked using a memory-safe, but not constant-time, wrapper around `strcmp()`

The above is one example out of several cases found, all of which are noted above.

Exploit Scenario
An application uses a libcurl build with TLS-SRP enabled and allows multiple users to make TLS connections to a remote server. An attacker times how quickly cURL responds to his requests to create a connection, and thereby gradually works out the credentials associated with an existing connection. Eventually, he is able to submit a request with exactly the same SSL configuration such that another user's existing connection is reused.

Recommendations

Short term, introduce a method, e.g. `Curl_constcmp()`, which does a constant-time comparison of two strings—that is, it scans both strings exactly once in their entirety.

Long term, compare secrets to user-submitted values using only constant-time algorithms.

5. Tab injection in cookie file

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-CURL-5
Target: `lib/cookie.c:895,984`	

Description

When libcurl makes an HTTP request, the cookie jar file is overwritten to store the cookies, but the storage format uses tabs to separate key pieces of information. The cookie parsing code for HTTP headers strips the leading and trailing tabs from cookie keys and values, but it does not reject cookies with tabs inside the keys or values.

In the snippet of `lib/cookie.c` below, `Curl_cookie_add()` parses tab-separated cookie data via `strtok_r()` and uses a switch-based state machine to interpret specific parts as key information:

```
firstptr = strtok_r(lineptr, "\t", &tok_buf); /* tokenize it on the TAB */
```

Figure 5.1: Parsing tab-separated cookie data via `strtok_r()`

Exploit Scenario

A webpage returns a `Set-Cookie` header with a tab character in the cookie name. When a cookie file is saved from cURL for this page, the part of the name before the tab is taken as the key, and the part after the tab is taken as the value. The next time the cookie file is loaded, these two values will be used.

```
% echo "HTTP/1.1 200 OK\r\nSet-Cookie: foo\tbar=\r\n\r\n\r\n"|nc -l 8000 &
% curl -v -c /tmp/cookies.txt http://localhost:8000
*   Trying 127.0.0.1:8000...
* Connected to localhost (127.0.0.1) port 8000 (#0)
> GET / HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.79.1
> Accept: */*
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
* Added cookie foo  bar="" for domain localhost, path /, expire 0
< Set-Cookie: foo   bar=
* no chunk, no close, no size. Assume close to signal end
```

Figure 5.2: Sending a cookie with name `foo\tbar`, and no value.

```
% cat /tmp/cookies.txt | tail -1
```

```
localhost    FALSE  /      FALSE  0      foo    bar
```

Figure 5.3: Sending a cookie with name `foo\tbar` and no value

Recommendations

Short term, either reject any cookie with a tab in its key (as `\t` is not a valid character for cookie keys, according to the relevant RFC), or escape or quote tab characters that appear in cookie keys.

Long term, do not assume that external data will follow the intended specification. Always account for the presence of special characters in such inputs.

6. Standard output/input/error may not be opened

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-CURL-6
Target: `src/tool_main.c:83-105`	

Description

The function `main_checkfds()` is used to ensure that file descriptors 0, 1, and 2 (stdin, stdout, and stderr) are open before `curl` starts to run. This is necessary to avoid the case wherein, if one of those descriptors fails to open initially, the next network socket opened by cURL may gain an FD number of 0, 1, or 2, resulting in what should be local input/output being received from or sent to a network socket instead. However, pipe errors actually result in the same outcome as success:

```
static void main_checkfds(void)
{
#ifdef HAVE_PIPE
  int fd[2] = { STDIN_FILENO, STDIN_FILENO };
  while(fd[0] == STDIN_FILENO || fd[0] == STDOUT_FILENO ||
        fd[0] == STDERR_FILENO || fd[1] == STDIN_FILENO ||
        fd[1] == STDOUT_FILENO || fd[1] == STDERR_FILENO)
    if(pipe(fd) < 0)
      return;   /* Out of handles. This isn't really a big problem now, but
                   will be when we try to create a socket later. */
  close(fd[0]);
  close(fd[1]);
#endif
}
```

Figure 6.1: `tool_main.c:83–105`, lines 83 through 105

Though the comment notes that an out-of-handles condition would result in a failure later on in the application, there may be cases where this is not true—e.g., the maximum number of handles has been reached at the time of this check, but handles are closed between it and the next attempt to create a socket. In such a case, execution might continue as normal, with stdin/out/err being redirected to an unexpected location.

Recommendations

Short term, use `fcntl()` to check if stdin/out/err are open. If they are not, exit the program if the pipe function fails.

Long term, do not assume that execution will fail later; fail early in cases like these.

7. Double free when using HTTP proxy with specific protocols

Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-CURL-7
Target: `curl/lib/url.c`, `curl/lib/http_proxy.c`	

Description

Using cURL with proxy connection and dict, gopher, LDAP, or telnet protocol triggers a double free vulnerability (figure 7.1). The `connect_init` function allocates a memory block for a `connectdata` struct (figure 7.2). After the connection, cURL frees the allocated buffer in the `conn_free` function (figure 7.3), which is freed for the second time in the `Curl_free_request_state` frees, which uses the `Curl_safefree` function on elements of the `Curl_easy` struct (figure 7.4).

This double free was also not detected in release builds during our testing — the glibc allocator checks may fail to detect such cases on some occasions. The two frees' success indicates that future memory allocations made by the program will return the same pointer twice. This may enable exploitation of cURL if the allocated objects contain data controlled by an attacker.

Additionally, if this vulnerability also triggers in libcurl—which we believe it should—it may enable the exploitation of programs that depend on libcurl.

```
$ nc -l 1337 | echo 'test' & # Imitation of a proxy server using netcat
$ curl -x http://test:test@127.0.0.1:1337 dict://127.0.0.1

2069694==ERROR: AddressSanitizer: attempting double-free on 0x617000000780 in thread T0:
  #0 0x494c8d in free (curl/src/.libs/curl+0x494c8d)
  #1 0x7f1eeeaf3afe in Curl_free_request_state curl/lib/url.c:2259:3
  #2 0x7f1eeeaf3afe in Curl_close curl/lib/url.c:421:3
  #3 0x7f1eeea30943 in curl_easy_cleanup curl/lib/easy.c:798:3
  #4 0x4e07df in post_per_transfer curl/src/tool_operate.c:656:3
  #5 0x4dee58 in serial_transfers curl/src/tool_operate.c:2434:18
  #6 0x4dee58 in run_all_transfers curl/src/tool_operate.c:2620:16
  #7 0x4dee58 in operate curl/src/tool_operate.c:2732:18
  #8 0x4dcf73 in main curl/src/tool_main.c:276:14
  #9 0x7f1eee2af082 in __libc_start_main
/build/glibc-SzIz7B/glibc-2.31/csu/../csu/libc-start.c:308:16
  #10 0x41c7cd in _start (curl/src/.libs/curl+0x41c7cd)

0x617000000780 is located 0 bytes inside of 664-byte region [0x617000000780,0x617000000a18)
freed by thread T0 here:
  #0 0x494c8d in free (curl/src/.libs/curl+0x494c8d)
  #1 0x7f1eeeaf6094 in conn_free curl/lib/url.c:814:3
  #2 0x7f1eeea92cc6 in curl_multi_perform curl/lib/multi.c:2684:14
```

```
  #3 0x7f1eeea304bd in easy_transfer curl/lib/easy.c:662:15
  #4 0x7f1eeea304bd in easy_perform curl/lib/easy.c:752:42
  #5 0x7f1eeea304bd in curl_easy_perform curl/lib/easy.c:771:10
  #6 0x4dee35 in serial_transfers curl/src/tool_operate.c:2432:16
  #7 0x4dee35 in run_all_transfers curl/src/tool_operate.c:2620:16
  #8 0x4dee35 in operate curl/src/tool_operate.c:2732:18
  #9 0x4dcf73 in main curl/src/tool_main.c:276:14
  #10 0x7f1eee2af082 in __libc_start_main
/build/glibc-SzIz7B/glibc-2.31/csu/../csu/libc-start.c:308:16

previously allocated by thread T0 here:
  #0 0x495082 in calloc (curl/src/.libs/curl+0x495082)
  #1 0x7f1eeea6d642 in connect_init curl/lib/http_proxy.c:174:9
  #2 0x7f1eeea6d642 in Curl_proxyCONNECT curl/lib/http_proxy.c:1061:14
  #3 0x7f1eeea6d1f2 in Curl_proxy_connect curl/lib/http_proxy.c:118:14
  #4 0x7f1eeea94c33 in multi_runsingle curl/lib/multi.c:2028:16
  #5 0x7f1eeea92cc6 in curl_multi_perform curl/lib/multi.c:2684:14
  #6 0x7f1eeea304bd in easy_transfer curl/lib/easy.c:662:15
  #7 0x7f1eeea304bd in easy_perform curl/lib/easy.c:752:42
  #8 0x7f1eeea304bd in curl_easy_perform curl/lib/easy.c:771:10
  #9 0x4dee35 in serial_transfers curl/src/tool_operate.c:2432:16
  #10 0x4dee35 in run_all_transfers curl/src/tool_operate.c:2620:16
  #11 0x4dee35 in operate curl/src/tool_operate.c:2732:18
  #12 0x4dcf73 in main curl/src/tool_main.c:276:14
  #13 0x7f1eee2af082 in __libc_start_main
/build/glibc-SzIz7B/glibc-2.31/csu/../csu/libc-start.c:308:16

SUMMARY: AddressSanitizer: double-free (curl/src/.libs/curl+0x494c8d) in free
```

Figure 7.1: Reproducing double free vulnerability with ASAN log

```
 158    static CURLcode connect_init(struct Curl_easy *data, bool reinit)
// (...)
 174        s = calloc(1, sizeof(struct http_connect_state));
```

Figure 7.2: Allocating a block of memory that is freed twice (curl/lib/http_proxy.c#158–174)

```
 787    static void conn_free(struct connectdata *conn)
// (...)
 814     Curl_safefree(conn->connect_state);
```

Figure 7.3: The `conn_free` function that frees the `http_connect_state` struct for HTTP CONNECT (curl/lib/url.c#787–814)

```
 2257    void Curl_free_request_state(struct Curl_easy *data)
 2258    {
 2259     Curl_safefree(data->req.p.http);
 2260     Curl_safefree(data->req.newurl);
```

Figure 7.4: The `Curl_free_request_state` function that frees elements in the `Curl_easy` struct, which leads to a double free vulnerability (curl/lib/url.c#2257–2260)

Exploit Scenario

An attacker finds a way to exploit the double free vulnerability described in this finding either in cURL or in a program that uses libcurl and gets remote code execution on the machine from which the cURL code was executed.

Recommendations

Short term, fix the double free vulnerability described in this finding.

Long term, expand cURL's unit tests and fuzz tests to cover different types of proxies for supported protocols. Also, extend the fuzzing strategy to cover `argv` fuzzing. It can be obtained using the approach presented in the `argv-fuzz-inl.h` from the AFL++ project. This will force the fuzzer to build an argv pointer array (which points to arguments passed to the cURL) from NULL-delimited standard input. Finally, consider adding a dictionary with possible options and protocols to the fuzzer based on the source code or on cURL's manual.

8. Some flags override previous instances of themselves

Severity: **Informational**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-CURL-8
Target: `src/tool_operate.c:1539`	

Description

Some cURL flags, when provided multiple times, overrides themselves and effectively use the last flag provided. If a flag makes cURL invocation's security options more strict, then accidental overwriting may weaken the desired security. The identified flag with this property is the `--crlfile` command-line option. It allows users to pass a PEM-formatted certificate revocation list to cURL.

```
       --crlfile <file>
              (TLS) Provide a file using PEM format with a Certificate Revocation
List that may specify peer certificates that are to be considered revoked.

              If this option is used several times, the last one will be used.

              Example:
               curl --crlfile rejects.txt https://example.com

              Added in 7.19.7.
```

Figure 8.1: The description of the `--crlfile` option

Exploit Scenario

A user wishes for cURL to reject certificates specified across multiple certificate revocation lists. He unwittingly uses the `--crlfile` flag multiple times, dropping all but the last-specified list. Requests the user sends with cURL are intercepted by a Man-in-the-Middle attacker, who uses a known-compromised certificate to bypass TLS protections.

Recommendations

Short term, change the behavior of `--crlfile` to append new certificates to the revocation list, not to replace those specified earlier. If backwards compatibility prevents this, have cURL issue a warning such as "`--crlfile specified multiple times, using only <filename.txt>`".

Long term, ensure that behavior, such as how multiple instances of a command-line argument are handled, is consistent throughout the application. Issue a warning when a security-relevant flag is provided multiple times.

9. Cookies are not stripped after redirect

Severity: **Low**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-CURL-9
Target: `--cookie flag`	

Description

If cookies are passed to cURL via the `--cookie` flag, they will not be stripped if the target responds with a redirect. RFC 9110 section 15.4, "Redirection 3xx", does not specify whether or not cookies should be stripped during a redirect; as such, it may be better to err on the side of caution and strip them by default if the origin changed. The recommended behavior would match the current behavior with cookie jar (i.e., when a server sets a new cookie and requests a redirect) and `Authorization` header (which is stripped on cross-origin redirects).

Recommendations

Short term, if backwards compatibility would not prohibit such a change, strip cookies upon a redirect to a different origin by default and provide a command-line flag that enables the previous behavior (or extend the `--location-trusted` flag).

Long term, in cases where a specification is ambiguous and practicality allows, always default to the most secure possible interpretation. Extend tests to check for behavior of passing data after redirection.

10. Use after free while using parallel option and sequences

Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-CURL-10
Target: `tool_operate.c:2251, tool_operate.c:2228, lib/sendf.c:275`	

Description

Using cURL with parallel option (-Z), two consecutive sequences (that end up creating 51 hosts), and an unmatched bracket triggers a use-after-free vulnerability (figure 10.1). The `add_parallel_transfers` function allocates memory blocks for an error buffer; consequently, by default, it allows up to 50 transfers (figure 10.2, line 2228). Then, in the `Curl_failf` function, it copies errors (e.g., `Could not resolve host: q{`) to appropriate error buffers when connections fail (figure 10.3) and frees the memory. For the last sequence (u~ host), it allocates a memory buffer (figure 10.2), frees a buffer (figure 10.3), and copies an error (`Could not resolve host: u~`) to the previously freed memory buffer (figure 10.4).

```
$ curl 0 -Z [q-u][u-~] }
curl: (7) Failed to connect to 0.0.0.0 port 80 after 0 ms: Connection refused
curl: (3) unmatched close brace/bracket in URL position 1:
}
 ^
curl: (6) Could not resolve host: q{
curl: (6) Could not resolve host: q|
curl: (6) Could not resolve host: q}
curl: (6) Could not resolve host: q~
curl: (6) Could not resolve host: r{
curl: (6) Could not resolve host: r|
curl: (6) Could not resolve host: r}
curl: (6) Could not resolve host: r~
curl: (6) Could not resolve host: s{
curl: (6) Could not resolve host: s|
curl: (6) Could not resolve host: s}
curl: (6) Could not resolve host: s~
curl: (6) Could not resolve host: t{
curl: (6) Could not resolve host: t|
curl: (6) Could not resolve host: t}
curl: (6) Could not resolve host: t~
curl: (6) Could not resolve host: u{
curl: (6) Could not resolve host: u|
curl: (6) Could not resolve host: u}
curl: (3) unmatched close brace/bracket in URL position 1:
}
 ^
====2789144==ERROR: AddressSanitizer: heap-use-after-free on address 0x611000004780 at pc
0x7f9b5f94016d bp 0x7fff12d4dbc0 sp 0x7fff12d4d368
WRITE of size 27 at 0x611000004780 thread T0
```

```
    #0 0x7f9b5f94016c in __interceptor_strcpy
../../../../src/libsanitizer/asan/asan_interceptors.cc:431
    #1 0x7f9b5f7ce6f4 in strcpy /usr/include/x86_64-linux-gnu/bits/string_fortified.h:90
    #2 0x7f9b5f7ce6f4 in Curl_failf /home/scooby/curl/lib/sendf.c:275
    #3 0x7f9b5f78309a in Curl_resolver_error /home/scooby/curl/lib/hostip.c:1316
    #4 0x7f9b5f73cb6f in Curl_resolver_is_resolved /home/scooby/curl/lib/asyn-thread.c:596
    #5 0x7f9b5f7bc77c in multi_runsingle /home/scooby/curl/lib/multi.c:1979
    #6 0x7f9b5f7bf00f in curl_multi_perform /home/scooby/curl/lib/multi.c:2684
    #7 0x55d812f7609e in parallel_transfers /home/scooby/curl/src/tool_operate.c:2308
    #8 0x55d812f7609e in run_all_transfers /home/scooby/curl/src/tool_operate.c:2618
    #9 0x55d812f7609e in operate /home/scooby/curl/src/tool_operate.c:2732
    #10 0x55d812f4ffa8 in main /home/scooby/curl/src/tool_main.c:276
    #11 0x7f9b5f1aa082 in __libc_start_main ../csu/libc-start.c:308
    #12 0x55d812f506cd in _start (/usr/local/bin/curl+0x316cd)

0x611000004780 is located 0 bytes inside of 256-byte region [0x611000004780,0x611000004880)
freed by thread T0 here:
    #0 0x7f9b5f9b140f in __interceptor_free
../../../../src/libsanitizer/asan/asan_malloc_linux.cc:122
    #1 0x55d812f75682 in add_parallel_transfers /home/scooby/curl/src/tool_operate.c:2251

previously allocated by thread T0 here:
    #0 0x7f9b5f9b1808 in __interceptor_malloc
../../../../src/libsanitizer/asan/asan_malloc_linux.cc:144
    #1 0x55d812f75589 in add_parallel_transfers /home/scooby/curl/src/tool_operate.c:2228

SUMMARY: AddressSanitizer: heap-use-after-free
../../../../src/libsanitizer/asan/asan_interceptors.cc:431 in __interceptor_strcpy
Shadow bytes around the buggy address:
  0x0c227fff88a0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c227fff88b0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c227fff88c0: fa fa fa fa fa fa fa fa fd fd fd fd fd fd fd fd
  0x0c227fff88d0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c227fff88e0: fd fd fd fd fd fd fd fd fa fa fa fa fa fa fa fa
=>0x0c227fff88f0:[fd]fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c227fff8900: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c227fff8910: fa fa fa fa fa fa fa fa fd fd fd fd fd fd fd fd
  0x0c227fff8920: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c227fff8930: fd fd fd fd fd fd fd fd fa fa fa fa fa fa fa fa
  0x0c227fff8940: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
Shadow byte legend (one shadow byte represents 8 application bytes):
  Heap left redzone:       fa
  Freed heap region:       fd
  ==2789144==ABORTING
```

Figure 10.1: Reproducing use-after-free vulnerability with ASAN log

```
 2192    static CURLcode add_parallel_transfers(struct GlobalConfig *global,
CURLM *multi, CURLSH *share, bool *morep, bool *addedp)
 2197    {
 // (...)
 2210   for(per = transfers; per && (all_added < global->parallel_max);
        per = per->next) {
 2227       if(!errorbuf) {
 2228         errorbuf = malloc(CURL_ERROR_SIZE);
 // (...)
 2249       result = create_transfer(global, share, &getadded);
```

```
2250        if(result) {
2251          free(errorbuf);
2252          return result;
2253        }
```

Figure 10.2: The `add_parallel_transfers` function (`curl/src/tool_operate.c#2192–2253`)

```
264     void Curl_failf(struct Curl_easy *data, const char *fmt, ...)
265     {
// (...)
275           strcpy(data->set.errorbuffer, error);
```

Figure 10.3: The `Curl_failf` function that copies appropriate error to the error buffer (`curl/lib/sendf.c#264–275`)

Exploit Scenario

An administrator sets up a service that calls cURL, where some of the cURL command-line arguments are provided from external, untrusted input. An attacker manipulates the input to exploit the use-after-free bug to run arbitrary code on the machine that runs cURL.

Recommendations

Short term, fix the use-after-free vulnerability described in this finding.

Long term, extend the fuzzing strategy to cover argv fuzzing. It can be obtained using the `argv-fuzz-inl.h` from the AFL++ project to build argv from stdin in the cURL. Also, consider adding a dictionary with possible options and protocols to the fuzzer based on the source code or cURL's manual.

11. Unused memory blocks are not freed resulting in memory leaks

Severity: **Low**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-CURL-11
Target: `tool_urlglob.c`, `tool_getparam.c`	

Description

For specific commands (figure 11.1, 11.2, 11.3), cURL allocates blocks of memory that are not freed when they are no longer needed, leading to memory leaks.

```
$ curl 0 -Z 0 -Tz 0
curl: Can't open 'z'!
curl: try 'curl --help' or 'curl --manual' for more information
curl: (26) Failed to open/read local data from file/application

=============2798000==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 4848 byte(s) in 1 object(s) allocated from:
    #0 0x7f868e6eba06 in __interceptor_calloc
../../../../src/libsanitizer/asan/asan_malloc_linux.cc:153
    #1 0x561bb1d1dc9f in glob_url /home/scooby/curl/src/tool_urlglob.c:459

Indirect leak of 8 byte(s) in 1 object(s) allocated from:
    #0 0x7f868e6eb808 in __interceptor_malloc
../../../../src/libsanitizer/asan/asan_malloc_linux.cc:144
    #1 0x561bb1d1e06c in glob_fixed /home/scooby/curl/src/tool_urlglob.c:48
    #2 0x561bb1d1e06c in glob_parse /home/scooby/curl/src/tool_urlglob.c:411
    #3 0x561bb1d1e06c in glob_url /home/scooby/curl/src/tool_urlglob.c:467

Indirect leak of 2 byte(s) in 1 object(s) allocated from:
    #0 0x7f868e6eb808 in __interceptor_malloc
../../../../src/libsanitizer/asan/asan_malloc_linux.cc:144
    #1 0x561bb1d1e0b0 in glob_fixed /home/scooby/curl/src/tool_urlglob.c:53
    #2 0x561bb1d1e0b0 in glob_parse /home/scooby/curl/src/tool_urlglob.c:411
    #3 0x561bb1d1e0b0 in glob_url /home/scooby/curl/src/tool_urlglob.c:467

Indirect leak of 2 byte(s) in 1 object(s) allocated from:
    #0 0x7f868e6eb808 in __interceptor_malloc
../../../../src/libsanitizer/asan/asan_malloc_linux.cc:144
    #1 0x561bb1d1dc6a in glob_url /home/scooby/curl/src/tool_urlglob.c:454
```

Figure 11.1: Reproducing memory leaks vulnerability in the `tool_urlglob.c` file with LeakSanitizer log.

```
$ curl 00 --cu 00
curl: (7) Failed to connect to 0.0.0.0 port 80 after 0 ms: Connection refused
```

```
=============2798691==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 3 byte(s) in 1 object(s) allocated from:
    #0 0x7fbc6811b3ed in __interceptor_strdup
../../../../src/libsanitizer/asan/asan_interceptors.cc:445
    #1 0x56412ed047ee in getparameter /home/scooby/curl/src/tool_getparam.c:1885

SUMMARY: AddressSanitizer: 3 byte(s) leaked in 1 allocation(s).
```

Figure 11.2: Reproducing a memory leak vulnerability in the `tool_getparam.c` file with LeakSanitizer log

```
$ curl --proto =0 --proto =0
Warning: unrecognized protocol '0'
Warning: unrecognized protocol '0'
curl: no URL specified!
curl: try 'curl --help' or 'curl --manual' for more information

================================================================
==2799783==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 1 byte(s) in 1 object(s) allocated from:
    #0 0x7f90391803ed in __interceptor_strdup
../../../../src/libsanitizer/asan/asan_interceptors.cc:445
    #1 0x55e405955ab7 in proto2num /home/scooby/curl/src/tool_paramhlp.c:385

SUMMARY: AddressSanitizer: 1 byte(s) leaked in 1 allocation(s).
```

Figure 11.3: Reproducing a memory leak vulnerability in the `tool_paramhlp.c` file with `LeakSanitizer` log

Exploit Scenario

An attacker finds a way to allocate extensive lots of memory on the local machine, which leads to the overconsumption of resources and a denial-of-service attack.

Recommendations

Short term, fix memory leaks described in this finding by freeing memory blocks that are no longer needed.

Long term, extend the fuzzing strategy to cover argv fuzzing. It can be obtained using the `argv-fuzz-inl.h` from the AFL++ project to build argv from stdin in the cURL. Also, consider adding a dictionary with possible options and protocols to the fuzzer based on the source code or cURL's manual.

12. Referer header is generated in insecure manner	
Severity: **Low**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-CURL-12
Target: `--referer` flag	

Description

The cURL automatically sets the `referer` header for HTTP redirects when provided with the `--referer ';auto'` flag. The header set contains the entire original URL except for the user-password fragment. The URL includes query parameters, which is against current best practices for handling the `referer`, which say to default to the `strict-origin-when-cross-origin` option. The option instructs clients to send only the URL's origin for cross-origin redirect, and not to send the header to less secure destinations (e.g., when redirecting from HTTPS to HTTP protocol).

Exploit Scenario

An user uses cURL to send a request to a server that requires multi-step authorization. He provides the authorization token as a query parameter and enables redirects with `--location` flag. Because of the server misconfiguration, a 302 redirect response with an incorrect `Location` header that points to a third-party domain is sent back to the cURL. The cURL requests the third-party domain, leaking the authorization token via the `referer` header.

Recommendations

Short term, send only the origin instead of the whole URL on cross-origin requests in the `referer` header. Consider not sending the header on redirects downgrading the security level. Additionally, consider implementing support for the `Referrer-Policy` response header. Alternatively, introduce a new flag that would allow users to set the desired referrer policy manually.

Long term, review response headers that change behavior of HTTP redirects and ensure either that they are supported by the cURL or that secure defaults are implemented.

References

- Feature: Referrer Policy: Default to strict-origin-when-cross-origin

13. Redirect to localhost and local network is possible (Server-side request forgery like)

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-CURL-13
Target: HTTP redirects	

Description
When redirects are enabled with cURL (i.e., the `--location` flag is provided), then a server may redirect a request to an arbitrary endpoint, and the cURL will issue a request to it. This gives requested servers partial access to cURL's users local networks. The issue is similar to the Server-Side Request Forgery (SSRF) attack vector, but in the context of the client application.

Exploit Scenario
An user sends a request using cURL to a malicious server using the `--location` flag. The server responds with a 302 redirect to `http://192.168.0.1:1080?malicious=data` endpoint, accessing the user's router admin panel.

Recommendations
Short term, add a warning about this attack vector in the `--location` flag documentation.

Long term, consider disallowing redirects to private networks and loopback interface by either introducing a new flag that would disable the restriction or extending the `--location-trusted` flag functionality.

14. URL parsing from redirect is incorrect when no path separator is provided

Severity: **Low**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-CURL-14
Target: URL parser	

Description

When cURL parses a URL from the `Location` header for an HTTP redirect and the URL does not contain a path separator ("/"), the cURL incorrectly duplicates query strings (i.e., data after the question mark) and fragments (data after cross). The cURL correctly parses similar URLs when they are provided directly in the command line. This behavior indicates that different parsers are used for direct URLs and URLs from redirects, which may lead to further bugs.

```
$ curl -v -L 'http://local.test?redirect=http://local.test:80?-123'
*   Trying 127.0.0.1:80...
* Connected to local.test (127.0.0.1) port 80 (#0)
> GET /?redirect=http://local.test:80?-123 HTTP/1.1
> Host: local.test
> User-Agent: curl/7.86.0-DEV
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 302 Found
< Location: http://local.test:80?-123
< Date: Mon, 10 Oct 2022 14:53:46 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
< Transfer-Encoding: chunked
<
* Ignoring the response-body
* Connection #0 to host local.test left intact
* Issue another request to this URL: 'http://local.test:80/?-123?-123'
* Found bundle for host: 0x6000039287b0 [serially]
* Re-using existing connection #0 with host local.test
* Connected to local.test (127.0.0.1) port 80 (#0)
> GET /?-123?-123 HTTP/1.1
> Host: local.test
> User-Agent: curl/7.86.0-DEV
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Mon, 10 Oct 2022 14:53:46 GMT
```

```
< Connection: keep-alive
< Keep-Alive: timeout=5
< Content-Length: 16
<
* Connection #0 to host local.test left intact
HTTP Connection!
```

Figure 14.1: Example logging output from cURL, presenting the bug in parsing URLs from the `Location` header, with port and query parameters

```
$ curl -v -L 'http://local.test?redirect=http://local.test%23-123'
*   Trying 127.0.0.1:80...
* Connected to local.test (127.0.0.1) port 80 (#0)
> GET /?redirect=http://local.test%23-123 HTTP/1.1
> Host: local.test
> User-Agent: curl/7.86.0-DEV
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 302 Found
< Location: http://local.test#-123
< Date: Mon, 10 Oct 2022 14:56:05 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
< Transfer-Encoding: chunked
<
* Ignoring the response-body
* Connection #0 to host local.test left intact
* Issue another request to this URL: 'http://local.test/#-123#-123'
* Found bundle for host: 0x6000003f47b0 [serially]
* Re-using existing connection #0 with host local.test
* Connected to local.test (127.0.0.1) port 80 (#0)
> GET / HTTP/1.1
> Host: local.test
> User-Agent: curl/7.86.0-DEV
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Mon, 10 Oct 2022 14:56:05 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
< Content-Length: 16
<
* Connection #0 to host local.test left intact
HTTP Connection!
```

Figure 14.2: Example logging output from cURL, presenting the bug in parsing URLs from `Location` header, without port and with fragment

Exploit Scenario

A user of cURL accesses data from a server. The server redirects cURL to another endpoint. cURL incorrectly duplicates the query string in the new request. The other endpoint uses the incorrect data, which negatively affects the user.

Recommendations

Short term, fix the parsing bug in the `Location` header parser.

Long term, use a single, centralized API for URL parsing in the whole cURL codebase. Expand tests with checks of parsing of redirect responses.

Summary of Recommendations

Trail of Bits recommends that cURL's developers address the findings detailed in this report and take the following additional steps in future versions of the software.

- Where it is possible to change cURL's behavior without breaking backwards-compatibility guarantees, homogenize the behavior of command-line flags to ensure that option-specific behavior does not catch users off-guard (see TOB-CURL-8).
- In cases where a protocol's specification is ambiguous, err on the side of the most secure interpretation or default, and provide an option to opt out if necessary.
- Implement the measures noted in the strategic fuzzing recommendations section above.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Documentation	The presence of comprehensive and readable codebase documentation
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Code Quality Recommendations

This appendix contains findings that do not have immediate or obvious security implications. However, they may facilitate exploit chains targeting other vulnerabilities or may become easily exploitable in future releases.

- **Redundant if-statement.** The second check for the value of `*connected` is unnecessary, perhaps the result of earlier code removals.

```
if(*connected) {
   /* ... omitted ... */
else {
  /* Add timeout to multi handle and break out of the loop */
  if(*connected == FALSE) {
     /* ... omitted ... */
  }
}
```

Figure C.1: A redundant check for the value of `*connected`.
(`curl/lib/ftp.c:508–524`)

- **`CONNECT` packet misnamed in comments.** Two comments refer to the MQTT `CONNECT` packet as the "CONN" packet, which could cause confusion. The MQTT spec itself, in section 3.1, uses "CONNECT" throughout.

```
/* add user to the CONN packet */
...
/* Set initial values of CONN packet */
```

Figure C.2: Comments misnaming MQTT packets.
(`curl/lib/mqtt.c:189,219`)

- **Outdated documentation.** The MQTT documentation at https://everything.curl.dev/usingcurl/mqtt#caveats states that cURL does not support MQTT authentication; however, cURL has since been updated to support it. The documentation should be updated to note this, and should also mention as a caveat that there is no TLS support and only MQTT v3 is supported (whereas v5 adds enhanced authentication support).

- **Missing deprecation information.** `CURLOPT_HTTPPOST` is not described as deprecated in a list of options for a curl easy handle. Also, the `CURLOPT_POSTFIELDS` document directs the reader toward `CURLOPT_HTTPPOST`.

D. HSTS debug patch

```
diff --git a/lib/http.c b/lib/http.c
index 531c6daf8..bdf9444cd 100644
--- a/lib/http.c
+++ b/lib/http.c
@@ -3715,7 +3715,14 @@ CURLcode Curl_http_header(struct Curl_easy *data, struct
connectdata *conn,
 #ifndef CURL_DISABLE_HSTS
   /* If enabled, the header is incoming and this is over HTTPS */
   else if(data->hsts && checkprefix("Strict-Transport-Security:", headp) &&
-          (conn->handler->flags & PROTOPT_SSL)) {
+          ((conn->handler->flags & PROTOPT_SSL) ||
+#ifdef CURLDEBUG
+           /* allow debug builds to circumvent the HTTPS restriction */
+           getenv("CURL_HSTS_HTTP")
+#else
+           0
+#endif
+            )) {
     CURLcode check =
       Curl_hsts_parse(data->hsts, data->state.up.hostname,
                       headp + strlen("Strict-Transport-Security:"));
```

Figure D.1: Patch to `http.c` that allows HSTS over HTTP in debug mode (to facilitate fuzzing).

E. Fix Review Results

On December 5, 2022, Trail of Bits reviewed the fixes and mitigations implemented by the cURL project team to resolve the issues identified in this report, as specified in the fix document at https://gist.github.com/bagder/6d9e94a3a90641b552cb26f44925b3e0.

In summary, the cURL project has sufficiently addressed eleven of the fourteen issues described in this report and accepted the risk associated with three.

We reviewed each fix to determine its effectiveness in resolving the associated issue. For additional information, please see the Detailed Fix Log.

ID	Title	Severity	Status
1	Bad recommendation in libcurl cookie documentation	Informational	Resolved
2	Libcurl URI parser accepts invalid characters	Undetermined	Resolved
3	Libcurl Alt-Svc parser accepts invalid port numbers	Undetermined	Resolved
4	Non-constant-time comparison of secrets	Low	Resolved
5	Tab injection in cookie file	Informational	Resolved
6	Standard output/input/error may not be opened	Informational	Resolved
7	Double free when using HTTP proxy with specific protocols	High	Resolved
8	Some flags override previous instances of themselves	Informational	Resolved
9	Cookies are not stripped after redirect	Low	Unresolved

10	Use after free while using parallel option and sequences	High	Resolved
11	Unused memory blocks are not freed resulting in memory leaks	Low	Resolved
12	Referer header is generated in insecure manner	Low	Unresolved
13	Redirect to localhost and local network is possible (Server-side request forgery like)	Informational	Unresolved
14	URL parsing from redirect is incorrect when no path separator is provided	Low	Resolved

Detailed Fix Review Results

TOB-CURL-1: Bad recommendation in libcurl cookie documentation
Resolved in PR #9654. The erroneous recommendation was removed.

TOB-CURL-2: Libcurl URI parser accepts invalid characters
Resolved in PR #9608 and PR #10096. The noted invalid characters are now rejected.

TOB-CURL-3: libcurl Alt-Svc parser accepts invalid port numbers
Resolved in PR #10095. The port number is now rejected as invalid if it begins with a non-digit character (e.g., a negative sign).

TOB-CURL-4: Non-constant-time comparison of secrets
Resolved in PR #9658. A new string comparison function `Curl_timestrcmp()` was introduced that has an execution time dependent only on the length of the shorter of its two input strings.

TOB-CURL-5: Tab injection in cookie file
Resolved in PR #9659. Cookies with tab characters in their name are now rejected entirely.

TOB-CURL-6: Standard output/input/error may not be opened
Resolved in PR #9663 and.#9708. The `main_checkfds()` function now calls `fnctl()` to check the status of `stdin`, `stdout`, and `stderr`, and will not continue until all three open without error.

TOB-CURL-7: Double free when using HTTP proxy with specific protocols
Resolved in commit 51c0ebcf; reported as CVE-2022-42915. We used dynamic testing to confirm that the vulnerability is no longer present.

TOB-CURL-8: Some flags override previous instances of themselves
Resolved in PR #9759. A new rule was introduced for contributors requiring that documentation for command line options specifically note how the option will be interpreted if the user supplies it multiple times.

TOB-CURL-9: Cookies are not stripped after redirect
Unresolved; risk accepted. *"This is by design and we do not see a way to change this behavior without breaking behavior for existing users. This is also not the typical way users use cookies with curl so those who do provide cookies like this usually do it in special crafted ways. This behavior is also documented in the man page for the --cookie option."*

TOB-CURL-10: Use after free while using parallel option and sequences
Resolved in PR #9729. We used dynamic testing to confirm that the vulnerability is no longer present.

TOB-CURL-11: Unused memory blocks are not freed resulting in memory leaks
Resolved in PR #9865. Dynamic testing revealed that the vulnerability noted in figures 11.1 (`tool_urlglob.c`), 11.2 (`tool_getparam.c`), and 11.3 (`tool_paramhlp.c`) are no longer present.

TOB-CURL-12: Referer header is generated in insecure manner
Unresolved; risk accepted. *"This topic was brought to the mailing list https://curl.se/mail/lib-2022-10/ 0039.html but no real desire in changing the default behavior was picked up, rather the opposite. This is a function that is rarely used, so presumably users that do not want this way of working will and should not enable it."*

TOB-CURL-13: Redirect to `localhost` and local network is possible
Unresolved; risk accepted. *"This is working as intended and I do not see how adding a warning here will help anyone or anything. The option asks curl to follow a redirect and there is nothing that says or indicates that it would not follow a redirect to certain hosts or host names. Of course it will also follow redirects to 'local' host names, for any definition of local."*

"The built-in redirect-following is not meant to be the way that covers all possible ways to do redirects for all users in all situations. If users need more control and more say in how redirects are done, curl and libcurl provide the means to let users follow them themselves. curl does not follow any redirects by default."

TOB-CURL-14: URL parsing from redirect is incorrect when no path separator is provided
Resolved in PR #9763. We used dynamic testing to confirm that the vulnerability is no longer present.