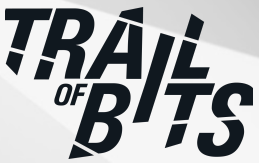


cURL

Threat Model

December 13, 2022

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

Prepared by: **Alex Useche and Anders Helsing**

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 80+ 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

Analysis Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and mutually 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 such, 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.

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 conduct a component-focused threat model of its cURL. From September 6 to October 7, 2022, a team of two consultants conducted a threat model of cURL and libcurl. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our assessment focused on the identification of security control flaws that could result in a compromise of confidentiality, integrity, or availability of the target system, especially with respect to the controls noted in the category breakdown table below. An exhaustive list of security control types and their definitions can be found in Appendix B.

Summary of Findings

The audit uncovered one design-level issue that could lead to vulnerabilities that compromise confidentiality, integrity, or availability of users and data handled by the system under audit.

FINDINGS BY SEVERITY

Severity	*Count*
High	1
Medium	3
Informational	3

FINDINGS BY CONTROL TYPE

Category	*Count*
System and Information Integrity	1
Awareness and Training	3
System and Communications Protection	1
Audit and Accountability	1
Configuration Management	1

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:

Alex Useche, Consultant
alex.useche@trailofbits.com

Anders Helsing, Consultant
anders.helsing@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 13, 2022	Status meeting #1
September 16, 2022	Interview meeting
September 20, 2022	Status meeting #2
September 27, 2022	Delivery of preliminary final report
October 11, 2022	Delivery of final report draft
December 13, 2022	Delivery of final report with fix review

Project Coverage

During a threat modeling assessment, engineers generally aim to cover the entire target system as a coherent whole. In some cases, however, certain components may be either unnecessary to examine, or impossible to review thoroughly.

Security Controls

The following security controls were used to evaluate the project targets during threat modeling exercises. Further information regarding security controls can be observed within Appendix B.

- Access Controls
- Audit and Accountability
- Awareness and Training
- Configuration Management
- Cryptography
- Denial of Service
- Identification and Authentication
- Risk Assessment
- System and Communications Protection
- System and Information Integrity

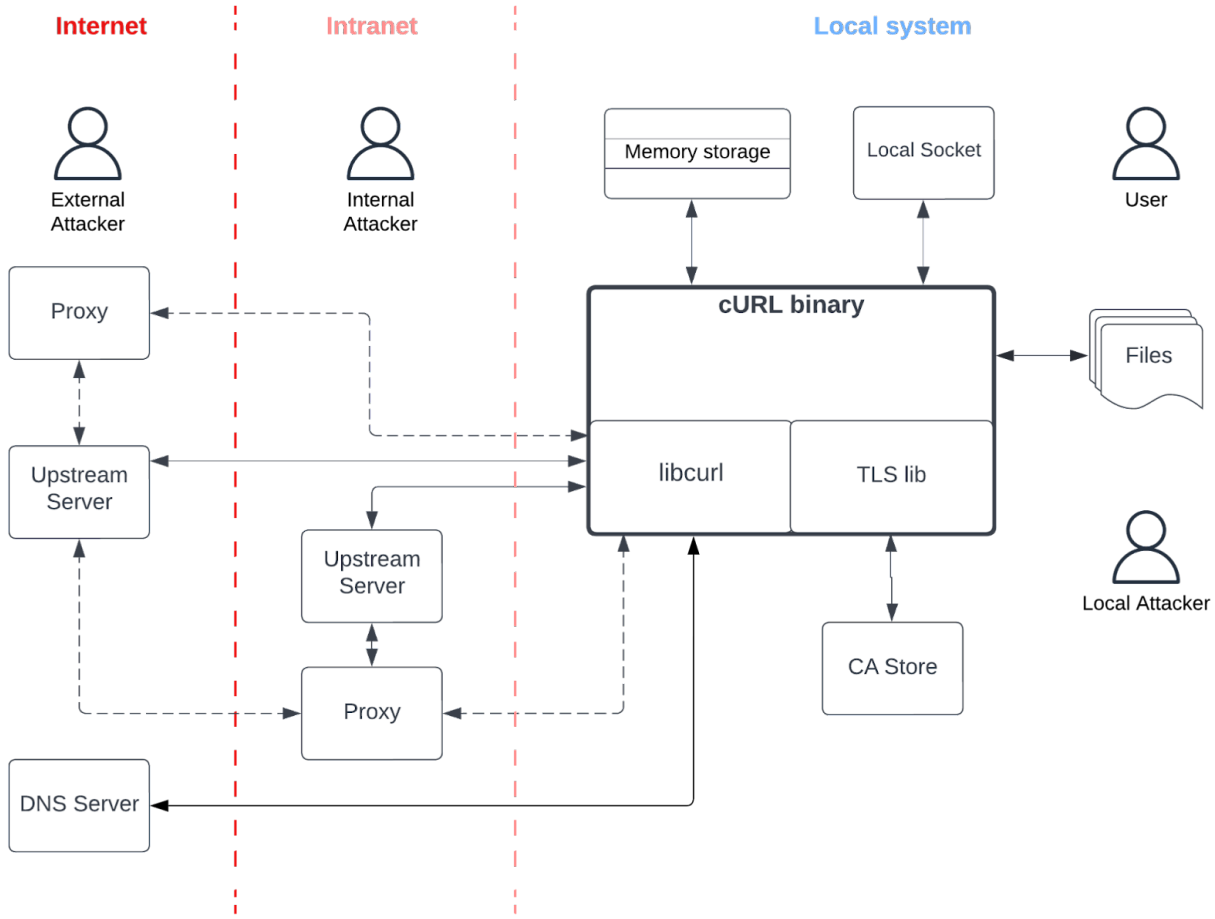
Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we focused on considering threats to cURL, including the command-line utility and libcurl. However, because cURL supports a long list of protocols, we chose not to treat each protocol implementation as a discrete component. As a result, our review considers support for various protocols at a high level while focusing on core components of cURL such as parsing, file input, output operations, and polling of connections.

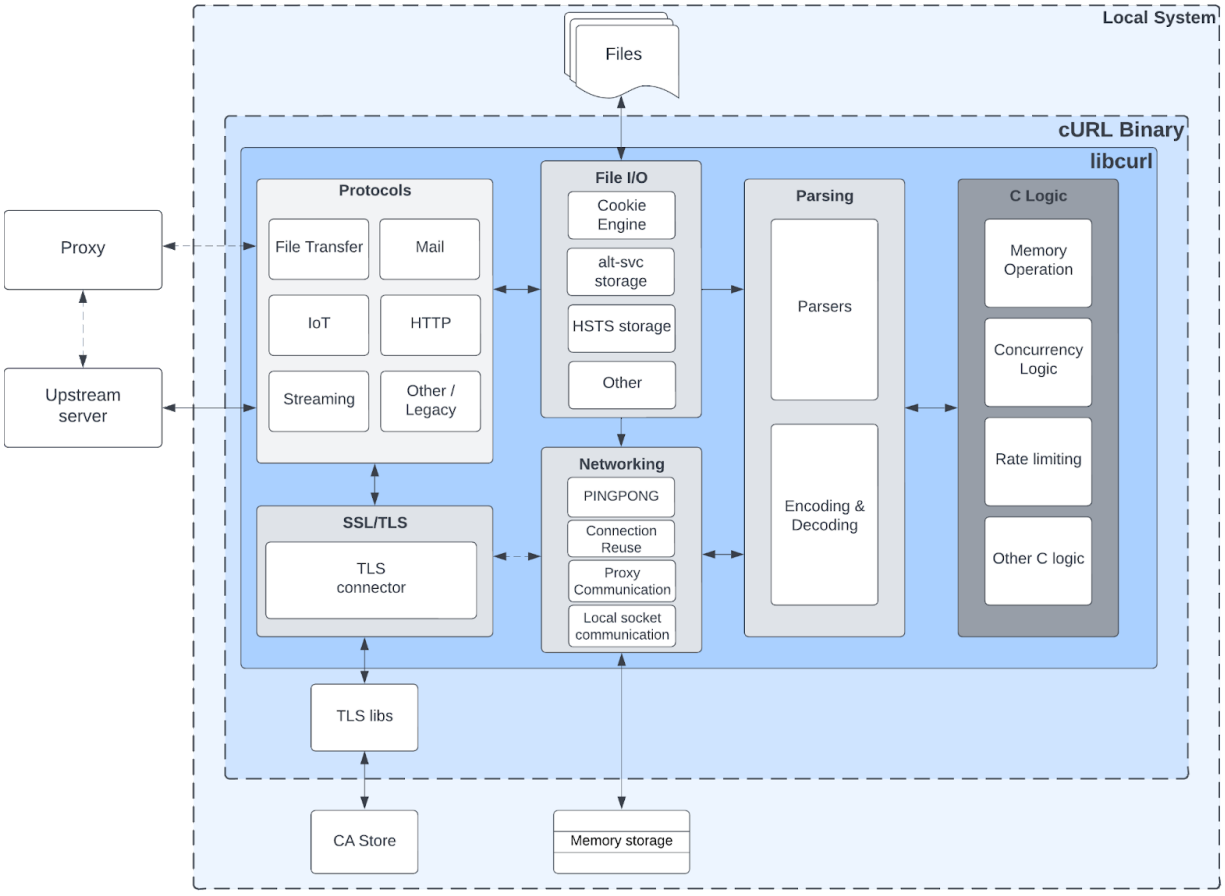
System Diagrams

The following diagrams depict the relationships between the target system's various components and trust zones, as well as the potential paths of threat actors within them.

High-Level Data Flow



Binary Data Flow



Components

cURL is a command-line utility and C library for data transfers with upstream servers over several supported protocols. The following tables describe the various components of cURL considered for the threat model.

High-Level Breakdown

Component	Description
cURL Binary	The binary uses libcurl and is compiled with a TLS library. This includes either cURL, the command line utility, or any application built with libcurl.
Upstream Server	The server with which cURL communicates (e.g., an HTTPS or FTP server). Can be located either on the internet or an intranet. May redirect communication to another upstream server.
Proxy	A proxy, either on the internet or an intranet, with which cURL is configured for making requests to the upstream server.
Memory storage	cURL uses in-process memory storage to save connections, TLS sessions, DNS responses, and other data.
Local file system	cURL uses the local file system to store and retrieve cookies, alternative service (alt-svc) information, HSTS entries, TLS certificates, logging output, environment variables, and other data. It can also load configuration files with options to use cURL.
libcurl	The core library on top of which cURL is built.
DNS server	Domain name resolution, with which cURL communicates for translating domain names to IP addresses and vice versa.
Local socker	Unix socket with which cURL can communicate via the `--unix-socket` flag.
CA Store	Used by the TLS library with which cURL is compiled. For some libraries (such as OpenSSL), cURL is responsible for iterating through the files in a CA store.

Binary Breakdown

Component	Description
Protocols	cURL supports several protocols that allow for two-way data transfers, most of which support URI schemes. We list protocols supported under the five categories listed below.
File Transfer	SCP, FILE, FTP, FTPS, SFTP, TFTP, SMB, SMBS
Mail	IMAP, IMAPS, POP3, POP3S, SMTP, SMTPS
HTTP	HTTP, HTTPS, WS, WSS
Streaming	RTSP, RTMP, RTMPS
IoT	MQTT
Other / Legacy	GOPHER, GOPHERS, LDAP, LDAPS, DICT, TELNET
SSL/TLS	Logic for handling in-transit encryption
TLS Connectors	TLS logic within the libcurl responsible for interacting with TLS libraries
TLS library	The TLS library with which cURL was compiled. cURL supports the following libraries: AmiSSL, BearSSL, BoringSSL, GnuTLS, libressl, mbedTLS, NSS, OpenSSL, rustls, Schannel, Secure Transport, and WolfSSL.
File I/O	Operations in cURL responsible for loading working with files stored on the system hard drive.
Cookie Engine	The cURL cookie engine keeps track of cookies for HTTP and HTTPS requests and uses the file system to load cookies and store cookie changes.
Alt-svc	Similar to the cookie engine, altsvc.c keeps track of atl-svc headers and loads them from local file storage.
HSTS	Similar to the cookie engine, hsts.c keeps track of HSTS header values and

	loads them from local file storage.
Other	cURL can also read environment files from memory, as well as other files for various operations, including • etags • TLS certificates • Configuration files such as `.netrc` and `.curlrc`
Networking	Logic responsible for establishing and maintaining connections to the various protocols.
PINGPONG	Generic back-and-forth support functions for certain protocols (e.g., FTP, IMAP, POP3, and SMTP).
Connection Reuse	The functionality for connection reuse via the connection cache.
Proxy communication	Logic for communicating via proxies.
DNS logic	Logic responsible for name resolution via flags like `--dns-servers` and `--doh-url`.
Local socket communication	Logic responsible for communicating with local sockets via `--unix-socket`.
Parsing	Logic used by cURL to parse request and response data.
Parsers	Parser logic in various APIs that are part of libcurl. For instance, the header API parses various request and response headers, and the URL API parses URLs used for various requests.
Encoders & Decoders	cURL's logic for encoding and decoding various data types, such as HTTP content types (deflate, gzip, zstd, and br) and chunked HTTP requests.
C Logic	Any other logic dealing in the cURL codebase that could lead to potential vulnerabilities, including unrestricted recursions and memory bugs.
Memory operations	Operations such as memory and buffer allocations.
Concurrency / Async	Operations responsible for calling operations asynchronously (e.g., making requests in a non-blocking manner).

Rate limiting	Operations that limit request rates.
Other C programming logic	Any other logic that could lead to bugs in the application.

Trust Zones

Systems include logical “trust boundaries” or “zones” in which components may have different criticality or sensitivity. Therefore, to further analyze a system, we decompose components into zones based on shared criticality rather than physical placement in the system. Trust zones capture logical boundaries where controls should or could be enforced by the system and allow designers to implement interstitial controls and policies between zones of components as needed.

Zone	Description	Included Components
Internet	The externally facing, wider internet zone. Components in this zone are untrusted.	• Upstream server • Proxy • DNS server
Intranet	Local network hosting the system running the cURL binary.	• Upstream server • Optional proxy
Local system	The local system running the cURL binary.	• cURL binary: Protocols, SSL/TLS, File I/O, Networking, Parsing, C logic, etc • Files • CA Store • Memory storage

Trust Zone Connections

At a design level, trust zones are delineated by the security controls that enforce the differing levels of trust within each zone. As such, it is necessary to ensure that data cannot move between trust zones without first satisfying the intended trust requirements of its destination. We enumerate such connections between trust zones below.

Originating Zone	Destination Zone	Data Description	Connection Type	Authentication Type
Local system	Internet	User data submitted via protocol specifications to the upstream server. Optionally, connection to proxy between server and cURL binary.	See Connection Type Breakdown DNS for name resolution	• Proxy authentication • See Connection Type Breakdown
Local System	Intranet	User data submitted via protocol specifications to the upstream server. Optional connection to proxy between server and cURL binary.	See Connection Type Breakdown	
Local System	Local System	cURL can make requests to Unix sockets using GET and POST requests.		
		cURL can generate C code by specifying a cURL command and using the `--libcurl` flag.	N/A	
		Local file IO for various tasks (see components table for a list), including configuration file reads.		
		Environment variables		

		are read by cURL, which could change its behavior. Environmental variables are read from memory and include settings that can in many cases be specified in configuration files instead.	
Local Network	Local System	Upstream servers on the local network with which cURL has established a connection will return data to a local system. Connections could be proxied via external or internal proxy servers.	See Connection Type Breakdown
Local Network	Internet	Proxy located in the intranet that sends data to an upstream server located on the internet. Data that was originally sent to a server in the intranet, but was redirected (fully or partially) to a server on the internet.	

Connection Type and Authentication Breakdown

The following table lists additional details regarding connection types and authentication methods available for supported protocols. For many protocols such as SMB, users can authenticate by sending credentials as part of the URI or request headers. However, we list only authentication types supported when cURL has a specific flag that users can pass to specify username and password and authentication types.

Protocol	Stateful	Stateless	TCP	UDP	Authentication
WS / WSS	X		X		-
RTSP	X		X		-
RTMP / RTMPS	X		X	X	-
MQTT	X		X		• Username and password
IMAP / IMAPS	X		X		• Username and password
POP3 / POP3S		X	X		• Username and password
SMTP / SMTPS		X	X		• Username and password
SCP	X		X		• Certificate authentication • Username and password
FTP / FTPS / SFTP	X		X		• Username and password • Certificate authentication (for SFTP) • Kerberos4 (for FTP) • kerberos5/GSSAPI (for FTP)

SMB / SMBS	X		X		• Username and password
LDAP / LDAPS	X		X	X	• Basic • NTLM • Digest
TELNET	X		X		-
HTTP / HTTPS		X	X		• Basic • Digest • NTLM • Negotiate • Bearer
FILE		X	X		-
TFTP		X		X	-
GOPHER / GOPHERS		X	X		-
DICT		X	X		-
FILE	N/A	N/A	N/A	N/A	-

Threat Actors

Similarly to establishing trust zones, defining malicious actors when conducting a threat model is useful in determining which protections, if any, are necessary to mitigate or remediate a vulnerability. We will use these actors in all subsequent findings from the threat model. Additionally, we define other "users" of the system who may be impacted by, or induced to undertake, an attack. For example, in a confused deputy attack such as cross-site request forgery, a normal user would be both the victim and the potential direct attacker, even though that user would be induced to undertake the action by a secondary attacker.

Actor	Description
External attacker	An attacker on the internet. They can control servers and proxies on the internet, eavesdrop, and create Man-in-the-Middle (MitM) connections in the internet space.
Internal attacker	An attacker on the intranet. They can eavesdrop and create MitM connections on the intranet. They cannot control either servers or proxies on the intranet.
Local attacker	An attacker sitting on the same machine where cURL application is being run. Has the same or lower level of privileges as the end user.
End user	A user who runs a built cURL binary.
libcurl user	Integrates libcurl in custom-developed applications.
Contributor	Regular contributor to the project.
Maintainer	A gatekeeper controlling additions to the project.
Malicious dependency developer	A source code dependency of curl that has been compromised.

Threat Actor Paths

Defining attackers' paths through the various zones is useful when analyzing potential controls, remediations, and mitigations that exist in the current architecture.

Originating Zone	Destination Zone	Actor	Description
Internet	Internet	Malicious dependency developer	Attackers can introduce malicious code in dependencies used in the cURL codebase, compromising users of the libcurl application and the cURL command line.
Internet	Intranet	External Attacker	Attackers will try to leak sensitive data intended only for intranet components. Similarly, they could target internal proxies used by cURL.
Internet	Local system	External Attacker	Attackers will attempt to manipulate data handled by cURL applications to gain access to the local system (e.g., by exploiting memory corruption bugs), to perform DoS attacks on the local system, or to infiltrate the system's internal network (e.g., via attacks similar to server-side request forgery).
Intranet	Intranet	Internal Attacker	Attackers will attempt to find and exploit bugs in cURL's network protocol implementations in order to bypass the protocols' security controls. Examples include dropping encryption from connections, downgrading protocol versions, impersonating authenticated connections, injecting data into established connections, and impersonating servers.

Local system	Local system	Local attacker	Attackers with access to a system running cURL will attempt local privilege escalation attacks by manipulating the local environment (variables, files, configurations, etc.) prior to end users' usage of the cURL application.

Possible Attack Vectors

At a high level, we consider the following non-exhaustive list of potential attack vectors based on the above analysis, in addition to the information gathered in our CVE analysis in Appendix C. Note that this list does not indicate there are vulnerabilities in each listed area; rather, it describes possible attack areas where attackers may look for points of failures and vulnerabilities to take advantage of.

- Invalid usage of libcurl by third-party application developers
- Flaws in protocols implementations, including:
  - Proxy communication
  - Non-conformance with protocol standards
  - Stateful vs. stateless protocol treatments
  - Authentication correctness issues
- Connection reuse flaws leading to issues such as CVE-2022-22576 and CVE-2022-27782
- Unexpected default behavior
- Flawed cross-endpoint transfers such as insufficient Same Origin Policy correctness and insecure HTTP redirects
- Flawed cross-protocol communication logic (e.g., redirects to other protocols, HTTP version changes, HTTP connection upgrades)
- Data transformations such as parsing, serialization, encoding, and data validation
- Memory safety issues
- Interaction with local system
- Insecure interaction with the kernel via networking interfaces or Unix sockets
- Incorrect or insecure DNS usage
- Race conditions and other concurrency bugs
- vTLS and TLS integration issues such as flaws in HSTS parsing or handling of certificates

Summary of Recommendations

Throughout the engagement, Trail of Bits identified a number of threat scenarios that may introduce risk within cURL. Trail of Bits recommends that the Linux Foundation address the findings detailed in this report and take the following additional steps to further build upon threat modeling exercises:

- Document deviations from RFCs and invariants for the various supported protocols, as a way to both inform cURL users of such deviations and document features that can be implemented or improved upon by cURL contributors. This can also drive property-based testing efforts.
- Consider implementing a property-based testing strategy driven by requirements specified by the RFCs for the various protocols supported by cURL.
- Devise a centralized input validation that relies on allowlists rather than denylists for checking against certain illegal characters per RFC specifications such as RFC 1738.
- Consider using tools such as weggli that allow you to write custom static analysis rules to run checks against potential issues, including non-conformance to RFC specification, insecure use of C functions such as `malloc`, and use of other insecure functions. These checks should run before merging pull requests into the codebase.
- Always default to secure settings for any operations that cURL performs. Implement terminal flags (such as `-k` for skipping certificate validation) to force users to tell cURL to skip or bypass secure defaults. See the recommendations in TOB-CURLTM-4 for a specific example.

Summary of Findings

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

ID	Title	Type	Severity
1	Proxy credentials are cached without encryption	System and Information Integrity	**Medium**
2	Lack of support for MQTT over TLS	System and Communications Protection	**Medium**
3	No warnings when TLS connection attempts fail with the --ssl flag	Audit and Accountability	**Informational**
4	Contributing guidelines lack recommendations against using insecure C functions	Awareness and Training	**Medium**
5	cURL treats localhost as secure by default	Configuration Management	**Informational**
6	Insufficient input validation strategy	Awareness and Training	**High**
7	Lack of documentation on supported protocol features and RFC compliance	Awareness and Training	**Informational**

Detailed Findings

1. Proxy credentials are cached without encryption	
Severity: **Medium**	Difficulty: **High**
Type: System and Information Integrity	Finding ID: TOB-CURLTM-1
Target: cURL	

Description
Users are able to configure cURL to communicate with the upstream server using a proxy they specify. cURL caches credentials for proxies provided by users in memory, so they can be reused for subsequent connections. These credentials are stored in memory in plaintext.

Threat Scenario
An attacker with access to the system running cURL uses a utility to dump heap memory used by cURL. The attacker then lists the credentials that the proxy uses to connect to the proxy and re-uses them to authenticate to it.

Justification
The severity is medium. Access to proxy credentials could allow the attacker to compromise the proxy.

The difficulty is high. Access to the system running cURL is required. Furthermore, the attacker would need to be able to run a utility to dump process memory from cURL.

Recommendations
Short term, consider encrypting proxy credentials in memory and clearing them as soon as they are no longer needed.

2. Lack of support for MQTT over TLS	
Severity: **Medium**	Difficulty: **High**
Type: System and Communications Protection	Finding ID: TOB-CURLTM-2
Target: cURL, libcurl	

Description
cURL supports data transfers and communication over the MQTT protocol. This protocol allows a client, such as a cURL, to communicate with an MQTT and subscribe to events for user-defined topics. However, communications with MQTT that are brokered over TLS (MQTTS) are not supported by cURL.

Threat Scenario
Eve decides to use cURL for data transfers over MQTT. She realizes that cURL does not support MQTTS, so she decides to instead communicate with the non-TLS MQTT broker. Her communications with the broker are now intercepted by an attacker in the network, resulting in a loss of confidentiality.

Justification
The severity is medium. Attackers with an MitM position will be able to read MQTT communications in plaintext.

The difficulty is high. Attackers would need to position themselves in the network and be able to capture communications between cURL and the upstream server. Furthermore, users are likely to use other utilities when they need to communicate with MQTTS servers.

Recommendations
Short term, extend cURL to support MQTT over TLS.

3. No warnings when TLS connection attempts fail with the `--ssl` flag

Severity: **Informational**	Difficulty: **High**
Type: Audit and Accountability	Finding ID: TOB-CURLTM-3
Target: cURL	

Description
cURL supports enabling TLS for various protocols with the `--ssl` flag. This flag tells cURL to communicate with the upstream server over TLS if the server supports it. In cases where cURL is unable to connect over TLS, data transfers will continue over plaintext or non-TLS connections without warning the user that the connection was not upgraded to TLS. Note that users can force the use of SSL by specifying the `--ssl-reqd` flag.

Threat Scenario
A user uses cURL to communicate with a server and specifies `--ssl`. The connection upgrade fails, and due to the lack of warning, the user believes the connection was established over TLS.

Recommendations
Short term, add a warning to STDOUT to notify users when connection upgrades fail. Consider adding a flag to allow users to ignore such warnings if they wish.

4. Contributing guidelines lack recommendations against using insecure C functions

Severity: **Medium**	Difficulty: **Medium**
Type: Awareness and Training	Finding ID: TOB-CURLTM-4
Target: cURL, libcurl	

Description
The cURL website includes documentation on contribution guidelines and a C style guide. However, neither document includes guidelines mandating or recommending secure C coding practices and standards, such as discouraging the use of insecure functions like `strcpy`, `atoi`, and `fscanf`. Although cURL uses `scripts/checksrc.pl` to disallow the use of certain functions, guidelines should exist that recommend secure C coding standards for contributors, particularly as some functions such as `strcpy` are not disallowed by the same script.

Threat Scenario
A developer pushes insecure code to cURL that is not caught by automated scripts or PR reviewers, introducing new vulnerabilities into the application or library.

Justification
The severity is medium. Insecure code and functions such as `atoi` and `strcpy` pushed to the codebase can introduce undefined behavior and other bugs that could lead to vulnerabilities such as code execution.

The difficulty is medium. The `scripts/checksrc.pl` script checks against the use of a list of banned functions listed in `docs/CHECKSRC.md`. Pull request requirements also reduce the likelihood that insecure code may be introduced.

Recommendations
Short term, include secure C coding guidelines in either the contribution guidelines or the C style guide. Either document should list the banned function that `scripts/checksrc.pl` checks against. If functions such as `scripts/checksrc.pl` continue to be avoidable by using non-standard libraries, the same document should describe how these functions should be used.

5. cURL treats localhost as secure by default

Severity: **Informational**	Difficulty: **High**
Type: Configuration Management	Finding ID: TOB-CURLTM-5
Target: cURL, libcurl	

Description

By default, cURL assumes that connection requests to `localhost`, `127.0.0.1`, and `[::1]` are secure and disables relevant security features, such as accepting the use of the `secure` cookie flag for insecure connections to `localhost` and cURL skipping name resolution checks. This may mislead cURL users into believing that their connections to `localhost` are secure.

Threat Scenario

A web developer uses cURL to make requests against a site they are developing and running on `http://localhost:8080`. Since cURL accepts and honors secure cookies from an insecure `localhost`, the developer assumes the application's behavior in `localhost` will match when it is deployed to production and makes assumptions about how the cookie flags will be treated when deploying to production.

Recommendations

Short term, explicitly document how cURL treats requests to `localhost` differently than requests to upstream servers.

Long term, update cURL so that it treats `localhosts` securely by default, and introduce a flag that users can use when calling cURL to turn off insecure behavior, such as disallowing cookies with the secure flag to be sent to `localhost` endpoints. This flag can work similarly to `-k`, which users can use when leveraging self-signed certificates to bypass validation.

6. Insufficient input validation strategy	
Severity: **High**	Difficulty: **Medium**
Type: Configuration Management	Finding ID: TOB-CURLTM-6
Target: cURL, libcurl	

Description
cURL performs input sanitization using a denylist of characters rather than strongly validating characters against an allowlist, regex, or similar. For instance, cURL allows potentially unsafe characters into cookie jar files, which could lead to broken functionality. This behavior deviates from relevant RFC specifications such as RFC 1738, which defines a set of permitted characters for URIs and disallows all others.

Threat Scenario
A zero-day exploit that takes advantage of weak URI validation is used against applications that rely on libcurl. Attackers leverage the exploit to compromise the confidentiality, integrity, or availability of user data and services that rely on such applications.

Justification
The severity is high. Because validation relies in many cases on denylists, it is difficult to account for future attacks that could make cURL vulnerable to attacks allowing malicious actors to compromise users, perform privilege escalation, or use cURL to run custom code remotely.

The difficulty is medium. There are no immediate concerns regarding allowed characters which cURL may not account for in their deny lists. However, deny lists are difficult to maintain and provide little protection against potential zero-day attacks, as new exploits may rely on the use of characters such as '`\t`', which cURL may not verify against.

Recommendations
Short term, default to using allow lists for sanitization and validation strategies for the various parsing tasks that cURL performs, such as cookie and URI parsing routines.

Long term, review RFCs for the various protocols and strings that cURL works with and parses and assure that the code conforms to the expectations outlined in such documents. Additionally, follow recommendations for TOB-CURLTM-6.

7. Lack of documentation on supported protocol features and RFC compliance

Severity: **Informational**	Difficulty: **High**
Type: Awareness and Training	Finding ID: TOB-CURLTM-7
Target: cURL, libcurl	

Description

cURL supports communications with upstream servers that rely on multiple protocols such as those listed in the Components table of this document. Data communications with each protocol must conform to the RFC documentation that is publicly available for each protocol. In some cases, conformance to RFC requirements is not strictly enforced by cURL for each protocol. Although upstream servers with which cURL communicates should enforce data conformance to applicable RFC requirements, attackers could take advantage of cURL's non-strict conformance to the same RFC for various attacks. Moreover, users of cURL, including application developers relying on libcurl, may incorrectly assume full compliance with the various protocol RFCs.

Threat Scenario

A developer makes assumptions about cURLs compliance to RFCs and implements a feature insecurely. An attacker notices this non-compliance and takes advantage of it to craft new attacks.

Recommendations

Short term, document deviations from RFCs and make it easily available for users of cURL so they understand where cURL stops data validation against RFC standards and when the responsibility is placed on upstream servers, developers, and users of cURL.

Long term, implement a property-based testing strategy that relies on testing specific properties defined in the RFC documents for every supported protocol.

A. Methodology

Trail of Bits's threat modeling assessments are intended to provide a detailed analysis of the risks facing an application at a structural and operational level, assessing the security of its design as opposed to its implementation details. During these assessments, engineers rely heavily on frequent meetings with the client's developers, paired with extensive readings of any and all documentation the client can make available. Code review and dynamic testing are not an integral part of threat modeling assessments, although engineers may occasionally consult the codebase or a live instance to verify specific assumptions about the system's design.

Engineers begin a threat modeling assessment by identifying the safeguards and guarantees that are critical to maintaining the target system's confidentiality, integrity, and availability. These *security controls* dictate the assessment's overarching scope, and are determined based on the specific requirements of the target system, which may include technical and reputational concerns, legal liability, regulatory compliance, and so on.

With these security controls in mind, engineers then divide the system into logical *components*—discrete elements that perform specific tasks—and establish *trust zones* around groups of components that lie within a common trust boundary. They identify the types of data handled by the system, enumerating the points at which data is sent, received, or stored by each component, as well as within and across trust boundaries.

Having established a detailed map of the target system's structure and data flows, engineers then identify *threat actors*—anyone who might threaten the target's security, whether a malicious external attacker, a naive insider, or otherwise. Based on each threat actor's initial privileges and knowledge, *threat actor paths* are then traced out through the system, establishing which controls and data a threat actor might be able to improperly access, as well as which safeguards stand in the way of such compromise. Any viable attack path discovered in this way constitutes a *finding*, which will also be paired with design recommendations by which such gaps in the system's defenses can be remediated.

After enumerating a list of findings, engineers rate the strength of each security control, indicating the general robustness of that type of defense against the full spectrum of possible attacks.

B. Security Controls and Rating Criteria

The following tables describe the security controls and rating criteria used in this report.

Security Controls for Threat Modeling assessment	
Category	**Description**
Access Controls	Authorization (including entitlement, access controls), session management, separation of duties, API and interfaces security, etc.
Audit and Accountability	Logging, non-repudiation, monitoring, analysis, reporting, etc.
Awareness and Training	Controls related to policies, procedures, and related capabilities
Configuration Management	Inventory, secure baselines, configuration management & change control
Cryptography	The cryptographic controls implemented at rest, in transit, and in-process
Denial of Service	The controls to defend against different types of denial-of-service attacks impacting availability
Identification and Authentication	User and system identification and authentication controls
Risk Assessment	Risk assessment policies, vulnerability scanning capabilities, and risk management solutions.
System and Communications Protection	Network level controls to protect data, network security, component security, and hardening, vendors' solutions and their integration, security of elements build internally
System and Information Integrity	Software integrity, malicious code protection, monitoring, information handling, and related controls

Rating Criteria	
Rating	**Description**

Strong	The security control was reviewed and no concerns were found.
Satisfactory	The security control had only minor issues; though it may lack certain non-critical operational procedures or security measures, their absence does not expose users to a significant degree of risk. Remediation in this area is suggested, but is not urgent.
Moderate	The security control had several issues or an impactful issue which may expose users to some degree of risk, albeit not to a severe degree. Remediation in this area is desired.
Weak	The security control had several significant issues which are likely to expose users to a substantial amount of risk. Remediation in this area should be prioritized.
Missing	The security control was found to be nonexistent or totally ineffective for its intended purpose, despite being necessary for the system's security. The implementation of this control should be prioritized.
Not Applicable	The security control is not applicable to this review.
Not Considered	The security control was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

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 threat is well known or common; an attacker can exploit it without significant effort or specialized knowledge.
Medium	An attacker must acquire in-depth knowledge of the system or expend a non-trivial amount of effort in order to exploit this issue.
High	An attacker must acquire complex insider knowledge or privileged access to the system in order to exploit this issue.

C. CVE Analysis

We analyzed the last CVEs (33) reported for cURL over the past three years. For each CVE, we examined the CWE to determine the key root cause and which aspects of the CIA triad were affected. Next, we determined which key components were affected by each CVE. This allowed us to better understand common attack paths from a historical perspective and to determine commonly affected components and root causes.

CVE	CWE	Root Cause	Impact	Component
CVE-2022-35252: control code in cookie denial of service	1286	Input validation (invalid server cookie accepted)	Availability	HTTP
CVE-2022-32208: FTP-KRB bad message verification	924	Data injection (injecting mitm data into error msg)	Integrity	FTP or KRB
CVE-2022-32207: Unpreserved file permissions	281	File permission problem (overwrite does not retain rights)	Confidentiality	“cookie.c”
CVE-2022-32206: HTTP compression denial of service	770	Unbounded compression chain from server	Availability	HTTP
CVE-2022-32205: Set-Cookie denial of service	770	Input validation (invalid server set-cookie:)	Availability	HTTP
CVE-2022-30115: HSTS bypass via trailing dot	319	Data validation (trailing dot can bypass HSTS)	Confidentiality	HSTS
CVE-2022-27782: TLS and SSH connection too eager reuse	295 840	Improper certificate Validation	Integrity	CONNECTION POOL
CVE-2022-27781:	835	Uncontrolled resource	Availability	URL, NSS

CERTINFO never-ending busy-loop	400	Consumption		
CVE-2022-27780: percent-encoded path separator in URL host	918 177	Improper handling of URL encoding	Integrity	URL
CVE-2022-27779: cookie for trailing dot TLD	668 201	Insertion of sensitive information into sent data	Confidentiality	URL
CVE-2022-27778: curl removes wrong file on error	706	Use of incorrectly-resolved name or reference	Integrity Availability	TOOL
CVE-2022-27776: Auth/cookie leak on redirect	522	Insufficiently protected credentials	Confidentiality	HTTP
CVE-2022-27775: Bad local IPv6 connection reuse	200	Exposure of sensitive information to an unauthorized actor	Confidentiality	CONNECTION POOL
CVE-2022-27774: Credential leak on redirect	522	Insufficiently protected credentials	Confidentiality	HTTP
CVE-2022-22576: OAUTH2 bearer bypass in connection re-use	287	Improper authentication	Confidentiality Integrity	CONNECTION POOL
CVE-2021-22947: STARTTLS protocol injection via MITM	345 310	Insufficient verification of data Authenticity / cryptographic Issues	Integrity	PINGPONG (used for several protocols)
CVE-2021-22946: Protocol downgrade required TLS bypassed	319 325	Missing cryptographic step	Confidentiality	PINGPONG (IMAP, POP3, FTP)

CVE-2021-22945: UAF and double-free in MQTT sending	415	Double free	Confidentiality	MQTT
CVE-2021-22926: CURLOPT_SSLCERT mixup with Secure Transport	295	Improper certificate validation	Availability	“DARWIN TLS”
CVE-2021-22925: TELNET stack contents disclosure again	457	Use of uninitialized variable	Confidentiality	TELNET
CVE-2021-22924: Bad connection reuse due to flawed path name checks	295	Improper certificate validation	Confidentiality	CONNECTION POOL
CVE-2021-22923: Metalink download sends credentials	522	Insufficiently protected credentials	Confidentiality	TOOL
CVE-2021-22922: Wrong content via metalink not discarded	20	Improper input validation	Integrity	TOOL
CVE-2021-22901: TLS session caching disaster	416	Use after free	Availability Integrity Confidentiality	CONNECTION POOL
CVE-2021-22898: TELNET stack contents disclosure	457	Use of uninitialized variable	Confidentiality	TELNET
CVE-2021-22897: schannel cipher selection surprise	488	Exposure of data element to wrong session	Confidentiality	SCHANNEL
CVE-2021-22890: TLS 1.3 session ticket proxy host	290	Authentication bypass by spoofing	Integrity	PROXY or VTLS

mixup				
CVE-2021-22876: Automatic referer leaks credentials	359	Exposure of private personal information to an unauthorized actor	Confidentiality	HTTP
CVE-2020-8286: Inferior OCSP verification	299	Improper check for certificate revocation	Integrity	OPENSSL
CVE-2020-8285: FTP wildcard stack overflow	674	Uncontrolled recursion	Availability	FTP
CVE-2020-8284: trusting FTP PASV responses	200	Exposure of sensitive information to an unauthorized actor	Confidentiality	FTP
CVE-2020-8231: wrong connect-only connection	825	Expired pointer dereference	Confidentiality	CONNECTION POOL
CVE-2020-8177: curl overwrite local file with -J	641	Improper restriction of names for files and other resources	Confidentiality	TOOL
CVE-2020-8169: Partial password leak over DNS on HTTP redirect	200	Exposure of sensitive information to an unauthorized actor	Confidentiality	HTTP

High-Level Analysis

CIA Triad Impact

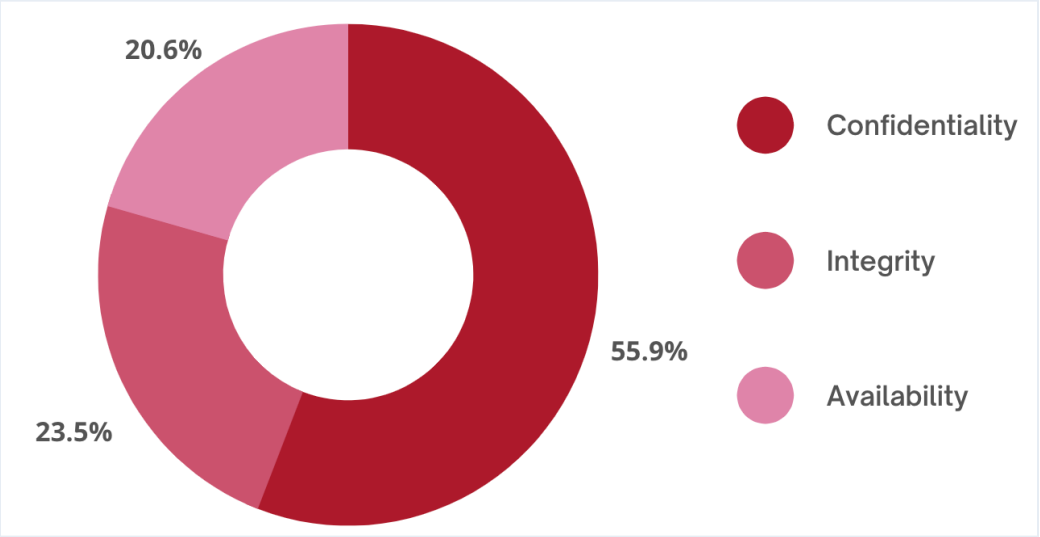


Figure C.1. CIA triad impact

Common CWEs

Count	CWE	Title
3	200	Exposure of Sensitive Information to an Unauthorized Actor
3	295	Improper Certificate Validation
3	522	Insufficiently Protected Credentials
2	319	Data validation (trailing dot can bypass HSTS)
2	457	Use of Uninitialized Variable
2	770	Allocation of Resources Without Limits or Throttling

Common Concerns

- Mishandling or ineffective use of TLS
- Revealing sensitive data in URL concerns by using TLS libraries ineffectively or incorrectly
- Caching of sensitive data via cookies (it is unclear how cURL handles cookies in requests and whether those are or can be cached)
- Uncontrolled recursions or infinite loops that cause code to hang

D. Fix Review Results

On December 6, 2022, Trail of Bits reviewed the fixes and mitigations implemented by the Linux Foundation, via OpenSSF and strategic partner Open Source Technology Improvement Fund team, to resolve the issues identified in this report. OpenSSF delivered fixes for some of the findings in this report, with associated pull requests when applicable.

In summary, Linux Foundation has sufficiently addressed one of the issues described in this report, partially resolved one, has not resolved two, and stated that they will not take action on three issues.

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	Proxy credentials are cached without encryption	Medium	Unresolved
2	Lack of support for MQTT over TLS	Medium	Unresolved
3	No warnings when TLS connection attempts fail with the --ssl flag	Informational	Resolved
4	Contributing guidelines lack recommendations against using insecure C functions	Medium	Unresolved
5	cURL treats localhost as secure by default	Informational	Partially resolved
6	Insufficient input validation strategy	High	Unresolved
7	Lack of documentation on supported protocol features and RFC compliance	Informational	Unresolved

Detailed Fix Review Results

TOB-CURLTM-1: Proxy credentials are cached without encryption
Unresolved. No changes were made at the time of this fix review.

TOB-CURLTM-2: Lack of support for MQTT over TLS
Unresolved. No changes were made at the time of this fix review.

TOB-CURLTM-3: No warnings when TLS connection attempts fail with the `--ssl` flag
Resolved. The cURL team added a warning to cURL as recommended in the reported finding (PR# 9519).

TOB-CURLTM-4: Contributing guidelines lack recommendations against using insecure C functions
Unresolved. The cURL team stated they will not be accepting the recommendation in the report.

TOB-CURLTM-5: cURL treats `localhost` as secure by default
Undetermined. The cURL team stated that they consider `localhost` to be secure. However, they added documentation to `docs/HTTP-COOKIES.md` to explicitly state how `localhost` is treated as secure by default by cURL (PR# 9938).

TOB-CURLTM-6: Insufficient input validation strategy
Unresolved. The cURL team stated they cannot fully accept the recommendation in the report, citing existing levels of strictness in testing parsers, including with fuzzers. The finding is unresolved as the input validation strategy continues to rely on denylists.

TOB-CURLTM-7: Lack of documentation on supported protocol features and RFC compliance
Unresolved. The cURL team stated that they will not address the recommendations provided for this finding and mentioned that all supported features are documented in depth, with details and examples. They also noted the infeasibility of documenting compliance with RFCs.