

Systems security in practice: Threat modelling

Kelly Kaoudis, kelly.kaoudis@trailofbits.com

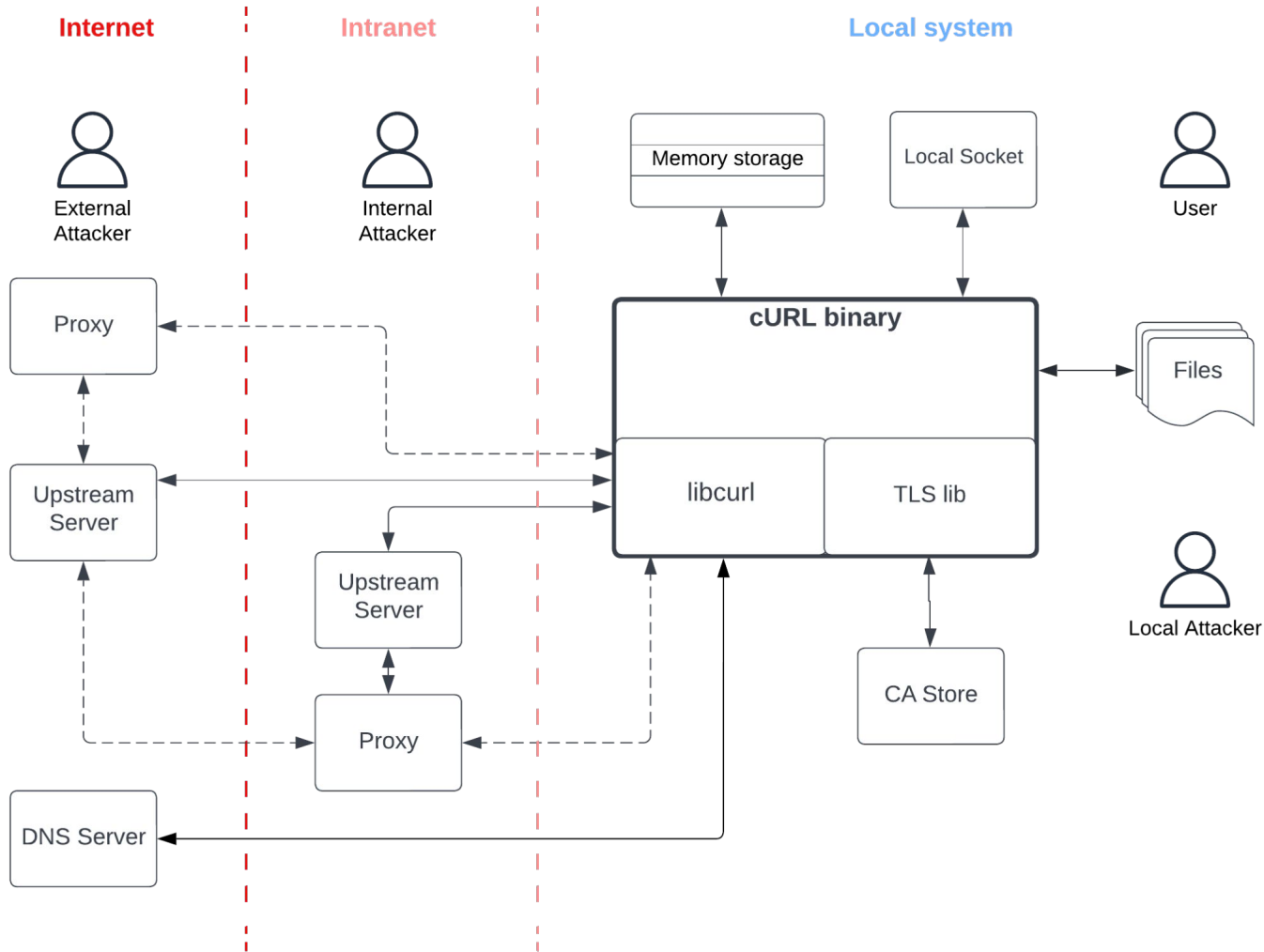
Agenda

- How we think about a system
 - Security controls
 - Whodunnit?
 - Scope and assumptions
 - Threats, risk, vulns, and findings
- Threat modelling in security research
 - Differences, similarities with industry threat models
 - Examples

Main idea: threat modelling informs and enables making good system-level security decisions!

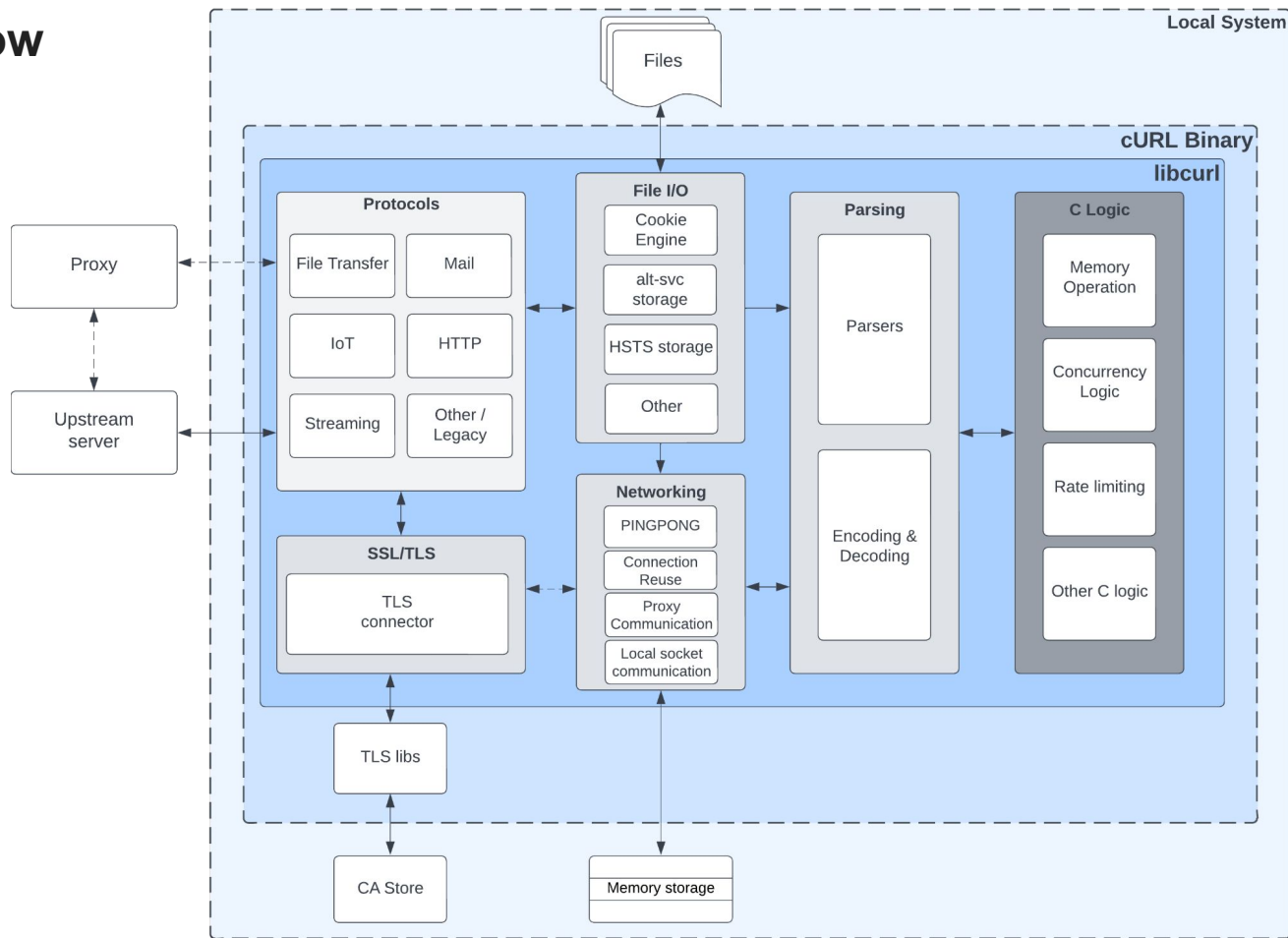
Curl Data Flow

Alex Useche, Anders Helsing



Curl Binary Data Flow

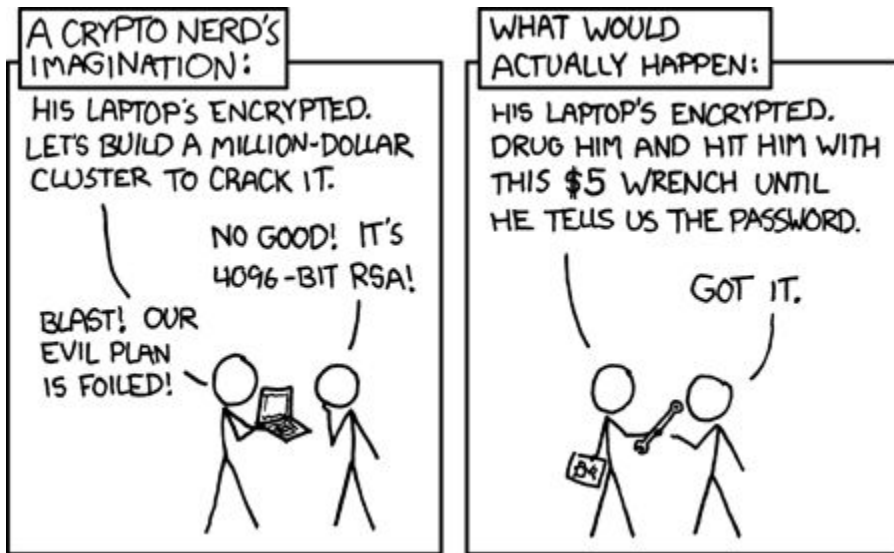
Alex Useche, Anders Helsing



Systems thinking



- Everything is interconnected (dependencies)
- Emergent properties
- Protections and countermeasures will layer
- Boundaries: input, output, exchange
- **Design-level** weaknesses and vulns



XKCD: Security, Source: <https://imgs.xkcd.com/comics/security.png>, License: [CC Attribution Noncommercial 2.5](https://creativecommons.org/licenses/by-nc/2.5/)

Security Controls

- Subset of categories of security **defences**
- Client agrees to the category list
- Maturity evaluation by category at audit end
- Example categories: Contingency Planning; System and Information Integrity; Auditing and Accountability

Voatz Security Controls Maturity Analysis

Stefan Edwards, Brian Glas

Planning (PL)	Missing	The system did not include plans for component interconnections, design intent, and the like.
Program Management (PM)	Not Applicable	Not assessed for this scenario.
Risk Assessment (RA)	Missing	The implementation team did not have a codified risk assessment process.
System and Communications Protection (SC)	Acceptable	The implementation team used strong, centrally managed TLS with certificate pinning in order to secure communications. However, the system used one wild-card certificate for all systems within the Voatz infrastructure, meaning that an attacker with access to one certificate could have complete access to all non-forwardly secret TLS communications.
System and Information Integrity (SI)	Weak	The system had minimal and diffuse controls surrounding the integrity of information stored therein. Notably, procedures, alerting, monitoring, and non-persistence were missing.
System and Services Acquisition (SA)	Acceptable	The implementation team acquired systems and services from reputable third parties, mainly Cloud C, Cloud A, and Cloud B.

“A vulnerability is any **trust assumption** involving people, processes, or technology that can be violated in order to exploit a system,”

NIST Special Publication 800-154

Mozilla's RRA (Rapid Risk Assessment)

- What does the system or application **do**?
- What **data** can it process or store?
- Confidentiality: What happens if all the data is disclosed to the world?
- Integrity: What if data is incorrect, misleading, website defaced, etc.?
- Availability: What if data or service is missing, deleted, unreachable?
- **Impact** (reputation, finances, productivity, system usability...)

PASTA: Process for Attack Simulation and Threat Analysis

1. Define objectives
2. Determine scope
3. Decompose the application into components, **trust boundaries**
4. Analyse **threats**
5. Analyse **vulnerabilities**
6. **Attack** analysis
7. Determine **risk** and **impact**



NIST SP 800-154: data-centric threat modelling

- Step 1: identify and characterise the **system** and **data** of interest
- Step 2: identify and select the **attack vectors** to be included in the model
- Step 3: characterise the **security controls** for mitigating the attack vectors
- Step 4: analyse the resulting threat model


Threat modelling at Trail of Bits!

1. Agree with client on initial scope
2. Discovery*: learn the system, actors, organization, process
3. Agree with client on assumptions
4. Show the gaps: threat scenarios, findings
5. Security controls maturity analysis
6. Report delivery and discussion

**refine scope; discuss each step of discovery with the client so they know (and expect) what we are doing*



Threat: motive, method

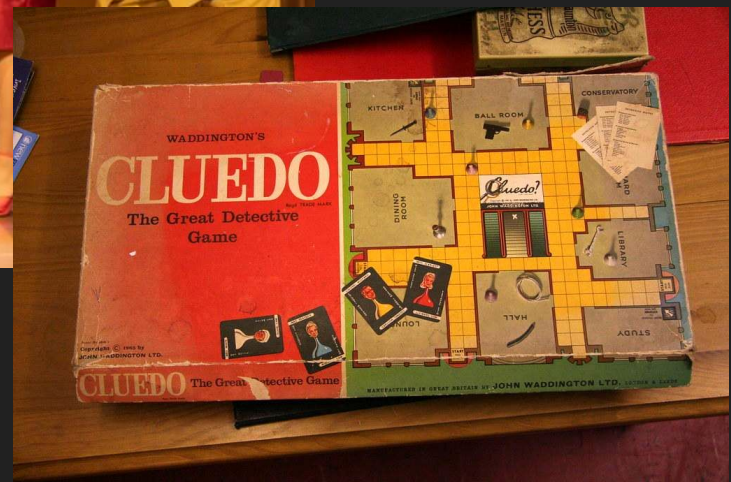
-  : a fault or error that results in unexpected output or behaviour
- **Vulnerability**: *incorrect assumption* of security where there is actually weakness (subclass of all bugs)
- **Threat** = motive (attacker's desire) + method (exploiting the vuln)

Who's in the system?

- Users, system admins or operators, attackers
- **What do they want?** Sensitive data, privileged access, persistence
- What *should* they be able to do?
- What *can* they do?
- What do they know?



Clue at the Citadel Theatre. Author: Nanc Price, Source: Alberta Prime Times.



Eclipse Jetty System Actors

Kelly Kaoudis, Spencer Michaels

External Attacker	An external attacker is an attacker on the public network (internet) from which at least one Jetty instance is accessible. This attacker can observe and analyze Jetty source commits as they land in the public repository for exploitable features.	Jetty Maintainer	A core Jetty contributor. Maintainers must review and approve pull requests prior to merging them.
Internal Attacker	An attacker on a private or application network from which at least one Jetty instance is accessible.	Application Developer	An application developer creates, maintains, and updates applications deployed via Jetty.
Client	“Client” refers to either a client of a Jetty server instance that can integrate the Jetty client libraries or a wholly distinct networked application.	Server Administrator	A server administrator administers a networked application that is either built with Jetty components, served via a Jetty instance embedded as a servlet container in another framework, or served via a standalone Jetty instance.
Local Attacker	A local attacker is an attacker who controls a process or user account on the same host as the Jetty instance and can affect the system environment, including the filesystem.	Server Deployer	A server deployer releases an application served via Jetty or built with Jetty components into the running environment. The deployer may not be a separate individual from the server administrator and application developer.
Jetty Contributor	A non-maintainer Jetty contributor.		

Components

<Component name>

<Component name> is a <noun> that <verb>s <specific types of data>. The <component name> can connect to <other component A> via <connection type specifics> and receives <specific data> from <another component B> over <connection type specifics>.

(Optionally, link to specific actor types that interact with the component)
<Actor> can <verb> the <component name>.

Eclipse JKube Components

Kelly Kaoudis

Watcher	The JKube Watcher allows for hot reloading of image configurations and resources.	Maven (*)	Maven is an open-source build tool for JVM languages. This component is out of scope.
Remote Dev	The JKube Remote Dev module allows developers to configure deployments for later remote SSH connections.	Cluster (*)	This is the remote cluster, orchestrated by Kubernetes or OpenShift. This component is out of scope.
Kit API	The Kit API is the standalone public API for building and deploying container images or configurations without using Maven or Gradle.	Docker, Dockerd (*)	Docker (and its daemon, dockerd) is a local container management system that can integrate with minikube (an optional Kubernetes component). This component is out of scope.
Docker API	The Docker API is a bespoke API that allows users to integrate JKube with dockerd directly or through a tool such as minikube.	Remote Registry (*)	JKube can push container images using delegated credentials to registries such as Docker Hub and Quay. Maven or Gradle can source packages from tools such as JFrog and Maven Central. This component is out of scope.

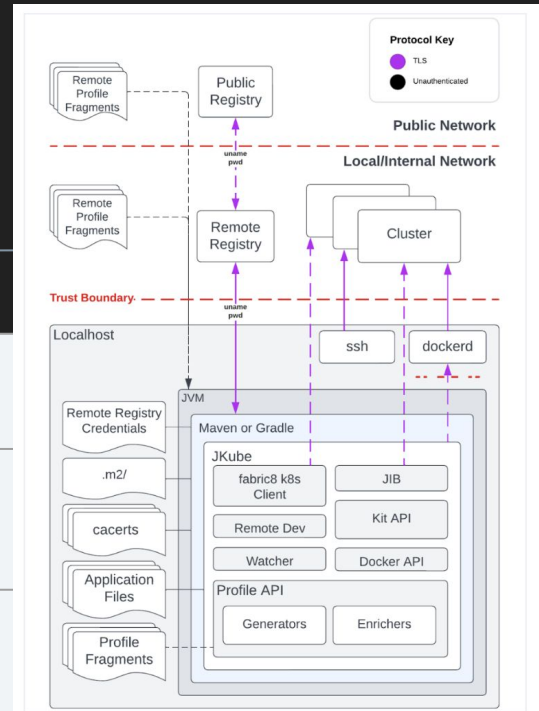
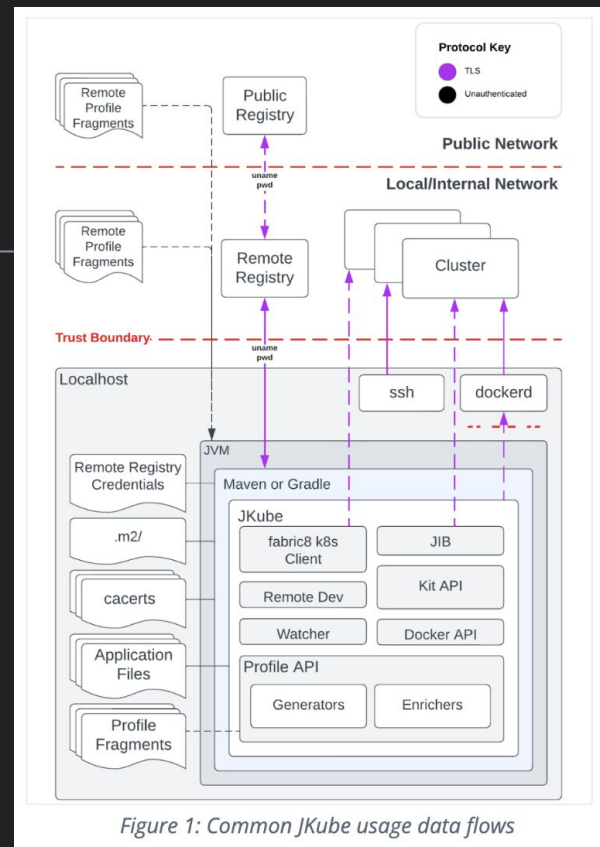


Figure 1: Common JKube usage data flows

Trust boundary

- **Adam Shostack:** “An attack surface is a trust boundary and a direction from which an attacker could launch an attack... a trust boundary is where entities with different privileges interact”
- **OWASP:** “A trust boundary is a location in data flow where level of trust changes”



Systematically thinking about threats: STRIDE

- Spoofing
- Tampering
- Repudiation
- Information disclosure
- Denial of service
- Expansion of authority

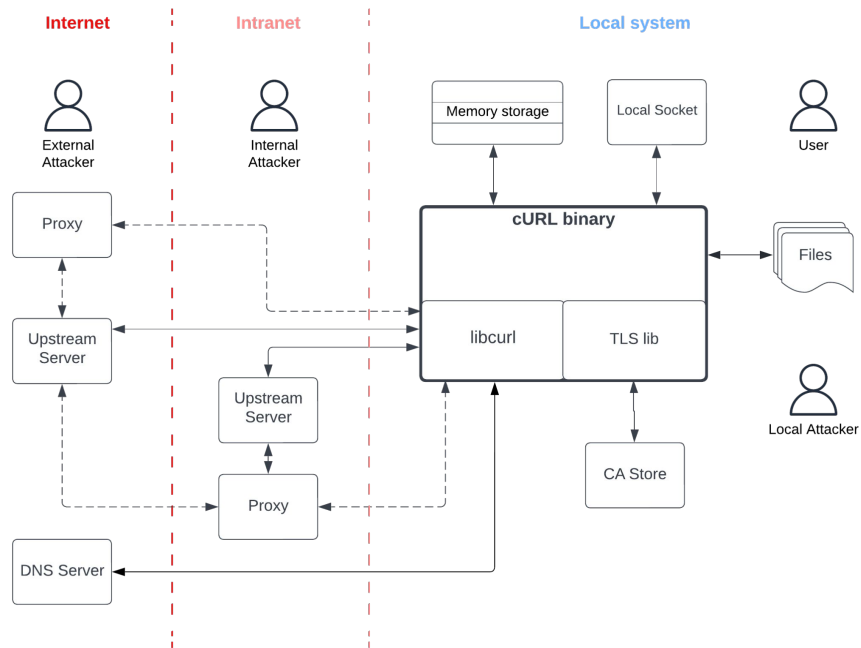
STRIDE: things that break user trust

- Spoofing violates authenticity
- Tampering violates integrity
- Repudiation violates non-repudiation
- Information disclosure violates confidentiality
- Denial of service violates availability
- Expansion of authority violates authorization (privilege enforcement)

Curl Threat Actor Paths

Alex Useche, Anders Helsing

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).



a finding = what if...

a gap + an actor + a vector

Severity	Difficulty
Informational	Undetermined
Undetermined	Low
Low	Medium
Medium	High
High	

risk = likelihood * impact

Low difficulty == high likelihood

Severity == impact

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.

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

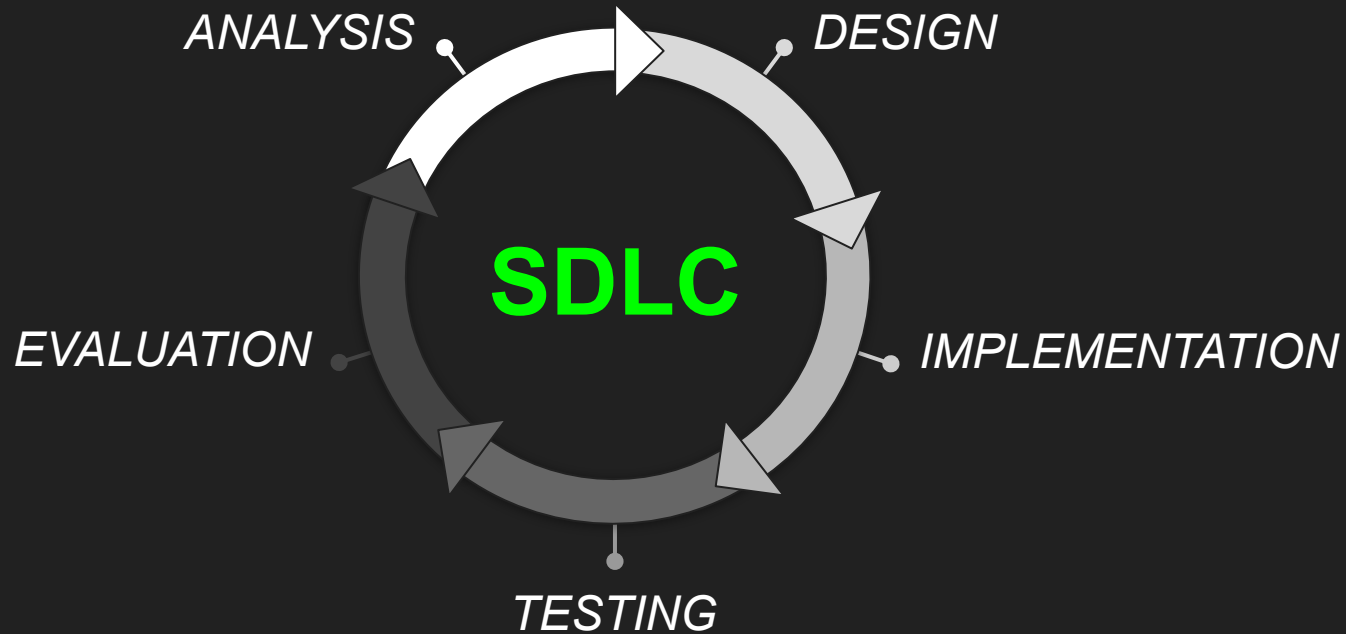
The severity is high. Because validation relies in many cases on denylists, it is difficult to account for future attacks that could make cURL vulnerable to attacks allowing malicious actors to compromise users, perform privilege escalation, or use cURL to run custom code remotely.

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](#).



“A threat model specifies the **conditions** under which a defense is designed to be secure and the **precise security guarantees** provided; it is an integral component of the defense itself... outlin[ing] what **type of actual attacker** the defense intends to defend against, guiding the evaluation of the defense... One of the defining properties of scientific research is that it is falsifiable: there must exist an experiment that can contradict its claims. Without a threat model, defense proposals are often either not falsifiable or trivially falsifiable,”

On Evaluating Adversarial Robustness, Carlini et al.

Writing a threat model for a paper

- Manage **scope**, e.g. “side channel attacks are out of scope”, “attackers with local hardware access are not considered”
- **Actors** (attackers, users) in the system
- **Capabilities** by actor, e.g., “we assume the administrator can access...”; “a remote attacker can... to escalate their privilege”; “a user knows...”
- Constrain **system** and environment
 - What components, actors, data can be protected by the prototype
 - What components, actors, data will be affected by the attack
 - Even... what emergent properties

(non-Trail) academic threat model example

“Improving Signal’s Sealed Sender”, NDSS 2021

I. Martiny, G. Kaptchuk, A. Aviv, D. Roche, E. Wustrow

C. Threat Model

We assume that the service provider (e.g. Signal) passively monitors messages to determine which pairs of users are communicating. This models either an insider threat or a service provider compelled to perform surveillance in response to a government request. We assume Alice and Bob have already exchanged delivery tokens and they communicate using sealed sender. Once initiated, we assume that Alice and Bob will continue to communicate over time. Finally, we also assume that many other users will be communicating concurrently during Alice and Bob’s conversation, potentially with Alice and/or Bob.

The service provider cannot view the contents of the encrypted sealed sender messages, but knows the destination user for these messages (e.g. someone sends a message to Bob). We assume that Alice and Bob have verified their respective keys out of band, and that the applications/devices they are using are secure. Although the service provider publishes the application, they typically distribute open-source code with deterministic builds, which we assume prevents targeting individual users.

We note that the service provider could infer a sender’s identity from network metadata such as the IP address used to send a sealed sender message. However, this is a problem that could be solved by using a popular VPN or an anonymizing proxy such as Tor [45], [19]. For the purposes of this paper, we assume that users who wish to remain anonymous to Signal can use such proxies (e.g. Orbot [2]) when sending sealed sender messages (and, in our solution, when receiving messages to ephemeral mailboxes), and we do not use network metadata in our attack.

In terms of impact, we note that a recent study suggests as many as 15% of mobile users already use VPNs every day [28]; this prevalence is even higher in east Asia and, presumably, among vulnerable user populations.

Here's one I worked on...

“Endoprocess: Programmable and Extensible Subprocess Isolation”, NSPW 2023

F. Yang, W. Huang, K. Kaoudis, A. Vahldiek-Oberwagner, N. Dautenhahn

2.2 Threat Model

We assume any component within a process may have exploitable vulnerabilities or be directly malicious. We assume that some operations (file operations, memory mappings, networking, etc.) can by design invoke privilege escalations [10] that bypass components' memory protections. An attacker can escalate their privilege through exploiting some component within a process. From this privilege escalation the attacker may gain read or write access to the entire process' memory space; or they may gain ability to launch data-only attacks without full control; or they may gain the ability to execute code on-behalf-of the system user that the current process runs as; or they may gain access to any system resources that the current user has authorization to access. The goal of an endoprocess is to prevent these intra-process privilege escalations.

The operating system, hardware platform, and monitor are part of the Trusted Computing Base, and assumed bug free and to have no backdoors. Side-channel attacks are considered out of scope. Finally, we assume developers can identify application components, annotate each component with the correct policy, and define appropriate boundaries, privilege restrictions, and data sharing policies.

Academic threat modeling

- Threat models in security papers
 - State assumptions
 - Contextualize with the who, where, and how of exploit(s)
 - Keep Reviewer 2's thought process on track!
- Systems thinking should always incorporate security!
- Scope of academic vs industry threat modeling goals may differ
- Trail of Bits is also happy to help ;)

Thank you!

Check out some of our work:
github.com/trailofbits/publications



Kelly Kaoudis
Senior Security Engineer
kelly.kaoudis@trailofbits.com