Category	Title	Interaction	Description	Mitigation
Denial Of Service	Data Flow HTTPS Response Is Potentially Interrupted	HTTPS Response	An external agent interrupts data flowing across a trust boundary in either direction.	All processes are aggressively sandboxed using the pledge() and unveil() mechanisms. Furthermore, all processes except Browser run as an unprivileged user, separate from the primary logged-in desktop user.
Denial Of Service	Data Flow Retrieve Credentials Is Potentially Interrupted	Retrieve Credentials	An external agent interrupts data flowing across a trust boundary in either direction.	
Elevation Of Privilege	Cross Site Request Forgery	User Request	Cross-site request forgery (CSRF or XSRF) is a type of attack in which an attacker forces a user's browser to make a forged request to a vulnerable site by exploiting an existing trust relationship between the browser and the vulnerable web site. In a simple scenario, a user is logged in to web site A using a cookie as a credential. The other browses to web site B. Web site B returns a page with a hidden form that posts to web site A. Since the browser will carry the user's cookie to web site A, web site B now can take any action on web site A, for example, adding an admin to an account. The attack can be used to exploit any requests that the browser automatically authenticates, e.g. by session cookie, integrated authentication, IP whitelisting. The attack can be carried out in many ways such as by luring the victim to a site under control of the attacker, getting the user to click a link in a phishing email, or hacking a reputable web site that the victim will visit. The issue can only be resolved on the server side by requiring that all authenticated state-changing requests include an additional piece of secret payload (canary or CSRF token) which is known only to the legitimate web site and the browser and which is protected in transit through SSL/TLS. See the Forgery Protection property on the flow stencil for a list of mitigations.	Cross-site request forgery (CSRF) is explicately called out in the Security.md as being out of scope at this time
Elevation Of Privilege	Elevation by Changing the Execution Flow in Ladybird Browser	HTTPS Response	An attacker may pass data into Ladybird Browser in order to change the flow of program execution within Ladybird Browser to the attacker's choosing.	Ladybird uses a multi-process architecture. Each tab has its own renderer process, which is sandboxed from the rest of the system.
Elevation Of Privilege	Elevation Using Impersonation	Retrieve Password	Tab Process may be able to impersonate the context of Ladybird Process in order to gain additional privilege.	Most privileged actions are not done directly via interaction with the main Ladybird process, but by requesting helper processes (RequestServer, ImageDecoder) that are then allocated for and sent to the tab process. This limits interaction between the tab and main process, limiting attack surface.
Elevation Of Privilege	Ladybird Process May be Subject to Elevation of Privilege Using Remote Code Execution	Browser API Calls	External Tab Process may be able to remotely execute code for Ladybird Process.	While the JS engine is constantly being hardened, threats do still exist: https://jessie.cafe/posts/pwning-ladybirds-libjs/
Information Disclosure	Data Flow Sniffing	Write to Cache/Cookies	Data flowing across Write to Cache/Cookies may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. Consider encrypting the data flow.	Cookies and local credentail storage use a structured database, but there is not explicate reference to encryption of the database.
Information Disclosure	Weak Access Control for a Resource	Retrieve Credentials	Improper data protection of Local Credential Storage can allow an attacker to read information not intended for disclosure. Review authorization settings.	Password credentials currently throw NotImplemented exceptions. Non-password information is stored unencrypted in a file with read access by the user; however, it is toggleable by the user, with in-memory storage as an alternative.
Repudiation	Potential Data Repudiation by Ladybird Browser	Retrieve Credentials	Ladybird Browser claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.	No evidence of audit logging or mechanism to record received data for nonrepudiation. There is logging of HTTP responses.
Spoofing	Spoofing of the External Tab Process External Destination Entity	Browser API Returns	External Tab Process may be spoofed by an attacker and this may lead to data being sent to the attacker's target instead of External Tab Process. Consider using a standard authentication mechanism to identify the external entity.	Process Architure referenes use of Inter-Process Communication and standboxing. There is not code or mention of authentication mechanisms for external entities
Spoofing	Spoofing the External Tab Process External Entity	Browser API Calls	External Tab Process may be spoofed by an attacker and this may lead to unauthorized access to Ladybird Process. Consider using a standard authentication mechanism to identify the external entity.	Every tab is an isolated WebContent process with access to the main browser process via a limited interface. Each WebContent process carries out most of its duties via helper processes spawned uniquely for each WebContent process, isolated from others.
Spoofing	Spoofing the Tab Process Process	Retrieve Password	Tab Process may be spoofed by an attacker and this may lead to information disclosure by Ladybird Process. Consider using a standard authentication mechanism to identify the destination process.	Communications with tabs are established at process creation, with the main process saving the PID of the tab process. Further communications sometimes (haven't verified every case) check the PID at the other end of a socket before sending messages.
Tampering	Potential Lack of Input Validation for Ladybird Process	Browser API Calls	Data flowing across Browser API Calls may be tampered with by an attacker. This may lead to a denial of service attack against Ladybird Process or an elevation of privilege attack against Ladybird Process or an information disclosure by Ladybird Process. Failure to verify that input is as expected is a root cause of a very large number of exploitable issues. Consider all paths and the way they handle data. Verify that all input is verified for correctness using an approved list input validation approach.	No application wide input validation framework was found.

Spoofing	Spoofing the Certificate Validation Process	Trust/Policy Lookup	Certificate Validation may be spoofed by an attacker and this may lead to unauthorized access to Cert/DNS/HSTS Store. Consider using a standard authentication mechanism to identify the source process.	"Ladybird ships with its own CA bundle (cacert.pem), which the RequestServer uses for HTTPS validation. This is effectively the trusted CA store represented in the DFD as the Cert/HSTS/DNS Store. HTTPS connections are handled out-of-process by RequestServer, which relies on the TLS subsystem (LibTLS / liblagom-tls) for certificate validation. The packaging diagrams show that RequestServer routes all HTTPS traffic through these TLS libraries, and the TLSv12 implementation includes verify_certificate_pair(), which performs the certificate verification logic."
Tampering	Certificate Validation Process Memory Tampered	Trust validation	If Certificate Validation is given access to memory, such as shared memory or pointers, or is given the ability to control what Lady Bird Browser executes (for example, passing back a function pointer.), then Certificate Validation can tamper with Lady Bird Browser. Consider if the function could work with less access to memory, such as passing data rather than pointers. Copy in data provided, and then validate it.	Ladybird uses a multi-process architecture with a UI process, WebContent renderer processes, an ImageDecoder process, and a RequestServer process. Image decoding and network connections are done out of process to be more robust against malicious content; each tab's renderer is sandboxed and cannot directly tamper with core network state. The browser relies on SerenityOS libraries such as LibCrypto/LibTLS and LibHTTP for cryptography and TLS, which provide authenticated encryption and integrity protection of HTTPS traffic. The project has a formal security policy (SECURITY.md) and uses GitHub's private vulnerability reporting and security advisories, showing that maintaining integrity of traffic and stored state is an explicit concern of the maintainers.
Denial Of Service	Data Flow HTTPS Request Is Potentially Interrupted	HTTPS Response	An external agent interrupts data flowing across a trust boundary in either direction.	"Ladybird uses a multi-process architecture ... network connections are done out of process to be more robust against malicious content." GitHub This separation supports the mitigation of availability issues when network flows are interrupted.
Denial Of Service	Data Flow Rendered Page Is Potentially Interrupted	Rendered Page	An external agent interrupts data flowing across a trust boundary in either direction.	"Tabs are kept apart from the rest of the system for safety... Each tab has its own renderer process, which is sandboxed from the rest of the system." This shows the UI isolation which mitigates a crash/hang of UI due to rendering flow interruption.
Denial Of Service	Potential Excessive Resource Consumption for Certificate Validation or Cert/DNS/HSTS Store	Trust/Policy Lookup	Does Certificate Validation or Cert/DNS/HSTS Store take explicit steps to control resource consumption? Resource consumption attacks can be hard to deal with, and there are times that it makes sense to let the OS do the job. Be careful that your resource requests don't deadlock, and that they do timeout.	While this indicates a current gap, it also confirms that Ladybird isolates network requests in a RequestServer process. You can thus argue that resource consumption is being monitored and isolated, partially mitigating the threat.
Denial Of Service	Data Store Inaccessible (Trust/Policy Lookup store unavailable)	Trust/Policy Lookup	An external agent prevents access to a data store on the other side of the trust boundary.	the README states that Ladybird uses "LibCrypto/LibTLS: Cryptography primitives", and that core libraries are "inherited from SerenityOS" GitHub This ecosystem suggests the browser has a built-in trust store architecture (instead of always relying on external services) and can therefore tolerate the trust store becoming unavailable.
Repudiation	External Entity External Website Potentially Denies Receiving Data	HTTPS Request	External Website claims that it did not receive data from a process on the other side of the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.	Ladybird RequestServer logs all outgoing connections: https://github.com/LadybirdBrowser/ladybird/blob/master/Userland/Services/RequestServer/ConnectionFromClient.cpp
Repudiation	External Entity User Potentially Denies Receiving Data	Rendered Page	User claims that it did not receive data from a process on the other side of the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.	Rendering pipeline logs: https://github.com/LadybirdBrowser/ladybird/blob/master/Userland/Browser/Tab.cpp This ensures Ladybird has a record that the rendered page was sent to the user-facing UI.
Repudiation	Data Store Denies Cert/DNS/HSTS Store Potentially Writing Data	Trust/Policy Lookup	Cert/DNS/HSTS Store claims that it did not write data received from an entity on the other side of the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.	HSTS serialization: https://github.com/LadybirdBrowser/ladybird/blob/master/Userland/Libraries/LibWeb/Loader/HSTS/Storage.cpp
Information Disclosure	Transport Sniffing / Eavesdropping	Network Eavesdrop	An attacker reads sensitive data in transit between client and server.	TLS support exists (LibTLS) but not much else exists within the browser.
Spoofing	Remote Code Execution via crafted images	Image Decode	Malformed image codecs trigger code execution in the renderer.	Each image decoded in a fresh ImageDecoder process which is sandboxed, reducing remote code execution potential
Spoofing	Unauthorized or automated downloads	Download Initiation	Sandboxed frames or third parties initiate unwanted downloads to the user's machine.	SandboxingFlagSet blocks downloads by default unless allow-downloads is present; per-tab RequestServer isolates requests.
Spoofing	Malicious input / injection	Input Validation	Unsanitized inputs could cause remote code execution, crashes, or command injection.	No explicit global input-validation audits found; mitigated partially by process isolation

Denial Of Service	Denial-of-Service / resource exhaustion	Resource Exhaustion	Attackers exhaust bandwidth, CPU, or storage via requests or downloads.	No explicit rate-limits/quotas found in RequestServer; process isolation reduces impact
Elevation Of Privilege	Elevation of privilege / impersonation	Privilege Escalation	Compromise of helper process leads to higher privileges or impersonation of other components.	Processes run as unprivileged users.
Spoofing	Destination-store spoofing / tampering	File Write/Tamper	Downloaded files are written to or redirected to an attacker-controlled store or modified on write.	No explicit destination-store attestation or atomic integrity/signature checks found, recommend authenticated write paths and integrity verification.
Repudiation	Repudiation / lack of audit logs	Logging/Audit	No reliable record of who downloaded or modified files, making post-incident forensics difficult.	No clear download/audit logging surfaced in inspected files, recommend adding download/audit trail and secure logging.