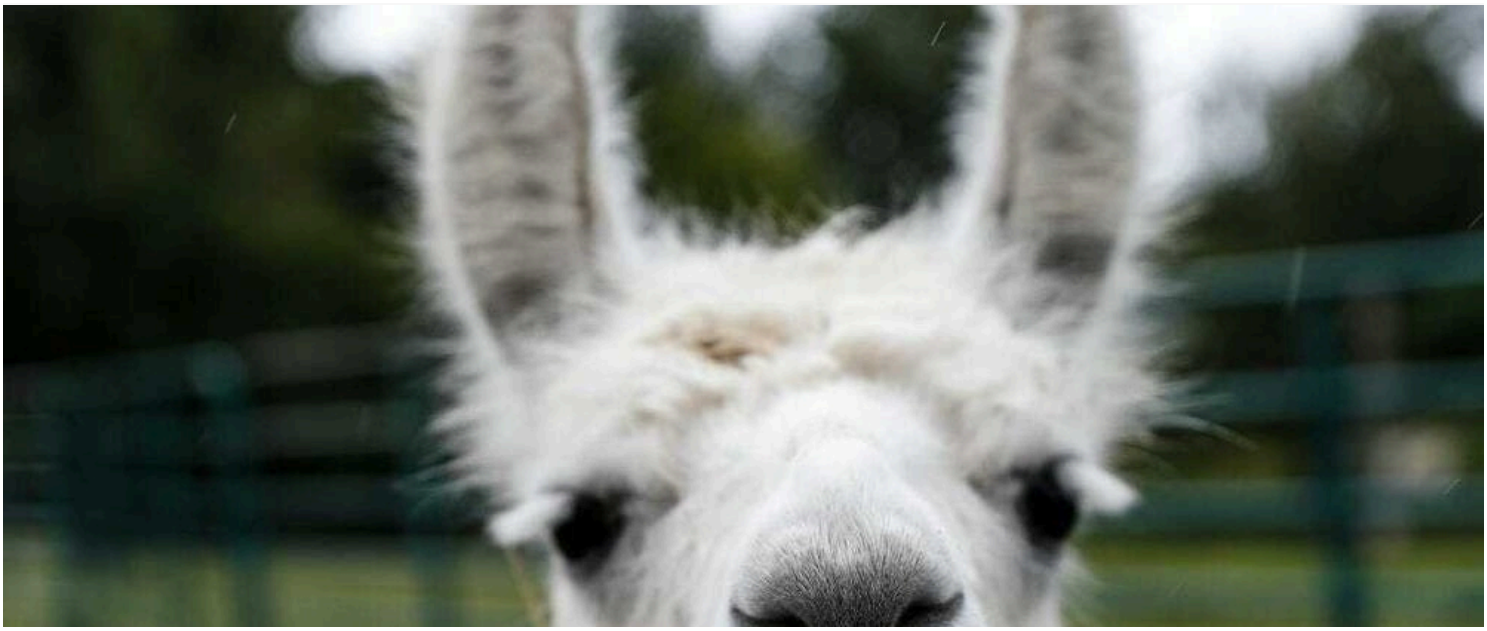

T
Posted on Jan 30 • Originally published at loopbreaker.substack.com

Exploring User Privacy in Ollama: Are Local LLMs Truly Private?

#ai #machinelearning #cybersecurity #opensource

Listen to this article on Substack

Ollama is a startup that offers an open-source tool for running large language models (LLMs) locally. Ollama has become the top choice for millions of users running AI locally, because of their focus and emphasis on user privacy.

Ollama was founded through the Y Combinator startup incubator—a fact that made my eyebrows twist into themselves in disbelief. Why would Y Combinator, a Silicon Valley incubator formerly run by Sam Altman, invest in a startup focused solely on user privacy? Why would they back a product that allows users to run LLMs privately?

While these questions remain unanswered for now, I went on a quest to find out: is Ollama truly private? Join me as I explore Ollama's code, behavior, and architecture.

I'll list my findings here. If you want to dive deeper into the details and hardening methods, go ahead and read the rest of the article.

#1 - After monitoring network usage and examining the code, I found no evidence of private user data being sent to an external server. However, I did find groundwork that might enable Ollama to do so in the future, without the users' knowledge or consent.

#2 - Ollama keeps the user's chat history in a plain text file named "history".

#3 - If the history file is deleted, Ollama creates it again silently.

#4 - Restricting access to the history file results in Ollama's refusal to load a model and start a session.

#5 - I've found a way to run Ollama without updating the history file, but the method wasn't disclosed in the documentation (see "Hardening Methods" below).

#6 - Ollama creates log files containing every interaction the user makes with the local API, including the name of the loaded model.

#7 - Ollama automatically updates itself from a remote server, there is no way to opt-out.

#8 - Pushing to an existing remote repository using Ollama will push all of the user's chat history along with it (according to a Youtube video, unverified).

Network Monitoring

The simplest way to verify whether a software is truly private is to check if it sends packets to an external server. Since Ollama is open-source, everybody's initial assumption is "If they were doing that, someone would have noticed by now"—which is generally true. However, I decided to verify myself rather than blindly trust them.

I wanted to poke around the source code and see if I could find anything out of place.

I started by going over Ollama's repository, which is written mostly in Golang, with some C++ for the llama.cpp engine. After getting familiar with the codebase, I found that Ollama is built as a local HTTP API. The CLI calls the local Ollama API, which handles model operations and data streaming.

The function *Host* is used to create the URL instance used for most API calls. It has "127.0.0.1" hardcoded as the default host for the URL—so far so good.

```
// config.go

func Host() *url.URL {

 ...

 if err != nil {
        host, port = "127.0.0.1", defaultPort
        if ip := net.ParseIP(strings.Trim(hostport, "[]")); ip != nil {
            host = ip.String()
        } else if hostport != "" {
            host = hostport
        }
 }

 ...

 return &url.URL{
        Scheme: scheme,
        Host:   net.JoinHostPort(host, port),
        Path:   path,
    }
}
```

I did find an automated external HTTP call to ***https://ollama.com/api/update*** under the function *IsNewReleaseAvailable*—which runs every hour and sends the user's operating system, machine architecture, current version, and timestamp to the server.

```
// updater.go

func IsNewReleaseAvailable(ctx context.Context) (bool, UpdateResponse) {

...

query := requestURL.Query()
query.Add("os", runtime.GOOS)
query.Add("arch", runtime.GOARCH)
query.Add("version", version.Version)
query.Add("ts", strconv.FormatInt(time.Now().Unix(), 10))
```

```
...

}
```

```
// index.ts (electron wrapper)

async function checkUpdate() {
    const available = await isNewReleaseAvailable()
    if (available) {
    logger.info('checking for update')
    autoUpdater.checkForUpdates()
  }
}

function init() {
  if (app.isPackaged) {
    checkUpdate()
    setInterval(() => {
      checkUpdate()
    }, 60 * 60 * 1000)
    }

  ...

}
```

Note that Ollama uses Electron's autoUpdater. According to Electron's documentation: "autoUpdater enables apps to automatically update themselves" (creepy). I couldn't find a way to opt out of automatic updates, meaning the code could potentially change at any minute, without user awareness or consent.

Network Usage

I could not find any obvious code blocks that send private user data out of the machine. I went a step further and verified network usage while using the software. I chatted with several LLMs for a few hours, while monitoring the software's network usage—and logged every network interaction from 8:00 PM to 3:00 AM.

```
ollama app.exe, Port: 15460 -> ollama.com @ 7:53 PM (Restart)
ollama app.exe, Port: 15460 -> ollama.com @ 8:53 PM down 5Kb, up 2Kb
ollama app.exe, Port: 15460 -> ollama.com @ 9:53 PM down 5Kb, up 2Kb
```

```
ollama app.exe, Port: 15460 -> ollama.com @ 10:53 PM down 5Kb, up 2Kb
ollama app.exe, Port: 15460 -> ollama.com @ 11:53 PM down 5Kb, up 2Kb
ollama app.exe, Port: 15460 -> ollama.com @ 12:53 AM down 5Kb, up 2Kb
ollama app.exe, Port: 15460 -> ollama.com @ 01:53 AM down 5Kb, up 2Kb
ollama app.exe, Port: 15460 -> ollama.com @ 02:53 AM down 5Kb, up 2Kb
```

This pattern matches the automated update from earlier. Notice the size of the packets as well—they would need more bandwidth to send chat history to an external server. It seemed quite promising so far, but there were still more paths to explore.

Log Files

By default, Ollama keeps detailed log files. I find it very dubious—why would a company so hyper focused on their user's privacy make their local logs so verbose by default?

What is the point of keeping detailed logs on a user's device by default, if they're not going anywhere?

These log files contain system information, but as far as I am concerned they do not contain inference history. They do, however, contain every interaction the user made with the local API, and the name of the model they have loaded.

```
[GIN] [REDACTED FOR PRIVACY] | 200 | 184.3285ms | 127.0.0.1 | POST "/api/chat"
[GIN] [REDACTED FOR PRIVACY] | 200 | 116.5207ms | 127.0.0.1 | POST "/api/chat"

llama_model_loader: loaded meta data with 36 key-value pairs and 197 tensors from
C:\\Users\\T\\.ollama\\models\\blobs\\sha256-(version GGUF V3 (latest))

llama_model_loader: - kv   0:            general.architecture str      = phi3
llama_model_loader: - kv   1:            general.type str              = model
llama_model_loader: - kv   2:            general.name str              = Phi 4
```

On Windows, you can find the logs by right-clicking Ollama's tray icon, then "View logs". On Linux, navigate to the following path.

```
Windows:
%appdata%/Local/Ollama/

Linux:
~/.ollama/logs
```

History File

The creepiest part about Ollama is its history file.

Ollama keeps all of the user's chat history, in plain text, stored in a file named "history". Why? I am not sure, but I have some ideas.

Imagine how surprised I was when I found the file contained all of my chat history, in plain text, for all to see. At this point questions kept popping up in my head—while I understand the need to keep some history on memory (not on file), why would it be stored in plain text? Why would there be a file in the first place? Why isn't any of this mentioned in Ollama's documentation? And why, when preventing access to the history file, does Ollama refuse to run?

Additionally, they could easily encrypt the file—and then have the software load it and use its content just the same. It doesn't have to be stored in plain text. An even better solution would be to not use a history file at all, and rely on history saved on memory during a session.

A few notes about Ollama's history file:

- During a session, Ollama creates a "history.tmp" file, where it keeps the current session's history in plain text. Once the session is closed, it aggregates the temporary buffer into the main history file and erases the temporary file.
- The history file has a hardcoded size limit, once a session is over if the history file is larger than the size limit—it removes the first chunk of the buffer until the file fits under the limit, essentially keeping the latest history.
- If the history file is deleted, Ollama creates it again silently. If the file is deleted during a session, Ollama creates it again silently and then copies the temporary file buffer into it.
- Making the history file read-only results in Ollama refusing to load a model or start a session.

Currently, Ollama does not send the history file to an external server (as far as I know, I might still be wrong). But considering the existence of such a file, combined with automatic updates the user can't opt out from—I can't help but think a silent update in the future might just change this behavior.

Just in case—I've gathered some hardening methods to make Ollama truly private.

Hardening Method #1

As straightforward as possible—simply block Ollama's network access through Windows firewall. On Linux, I believe it is possible to use UFW for this.

Hardening Method #2

Run Ollama with ***OLLAMA_NOHISTORY=1*** as an environment variable.

I've found this method while looking at Ollama's source code and verified it works. While not disclosed in the documentation (I wonder why?), it prevents writing to the history file. Note that this method's effectiveness could change if Ollama has network access. I recommend combining this method alongside method #1 to ensure privacy. Note that log files (under "View logs") will still be written to and updated.

Hardening Method #3 (Linux)

A workaround to prevent Ollama from writing to the history file on Linux can be found here, quoting Github user "glassbell".

> This workaround works for me (after deleting the history file with all of the warcrimes in it):
>
> ```
> $ touch history
> $ touch history.tmp
> \# chattr +i history
> \# chattr +i history.tmp
> ```
>
> (yes, it also tries to write to the second file)
>
> (also, I don't know how it does it, but if the i attribute is not set, ollama can recreate the file even if the owner is root and no permissions. Creepy)

I was not able to replicate the same behavior as ***chattr +i*** on Windows. Making the history file read-only causes Ollama to refuse to load models or start sessions. If you know of an alternative solution for Windows, please let me know!

How I Cut 22.3 Seconds Off an API Call with Sentry 🕐

Struggling with slow API calls? Dan Mindru walks through how he used Sentry's new Trace View feature to shave off 22.3 seconds from an API call.

Get a practical walkthrough of how to identify bottlenecks, split tasks into multiple parallel tasks, identify slow AI model calls, and more.

Read more →

Top comments (1)

Andrey Rusev • Jan 30

BTW - somebody I know on another network just published a link to a 'comprehensive 298-page International AI Safety Report':

gov.uk/government/publications/int...

According to the description:

> A report on the state of advanced AI capabilities and risks – written by 100 AI experts including representatives nominated by 33 countries and intergovernmental organisations.

If you feel like it - take a look... Might contain a lot of things we already suspect ;) (jokingly, of course:)

T

Cybersecurity Researcher. Writes Loopbreaker, a mix of my OCD-fueled musings about AI, Crypto and Cybersecurity.

LOCATION
Low Earth Orbit

EDUCATION
Self taught

JOINED
Nov 11, 2023

More from T

AI, but at What Cost? Breakdown of AI's Carbon Footprint
#ai #development #discuss #machinelearning

See Article