Secrets are forever: Characterizing sensitive file leaks on IPFS

Zhengyu Wu, Brian Kondracki, Nick Nikiforakis, Aruna Balasubramanian
Stony Brook University
Stony Brook, U.S.A
Emails: {zhenwu, bkondracki, nick, arunab}@cs.stonybrook.edu

***Abstract*—The InterPlanetary File System (IPFS) is an emerging peer-to-peer hypermedia protocol designed to enhance the speed, security, and openness of the web. Utilizing content-based addressing, IPFS establishes a decentralized, distributed, and trustless network for data storage and delivery. Despite its growing popularity, the inherent openness of IPFS raises concerns about accidental sharing of sensitive files, posing potential threats to user privacy and security. In this paper, we conduct a measurement study to investigate the extent of sensitive file sharing on the IPFS network.**

Using IPFS-search, a widely-used search engine indexing IPFS content, we identified over 2,000 files containing sensitive information such as API keys and private SSH keys. However, as IPFS-search operates on a centralized infrastructure, access restrictions may limit opportunistic attacks. To demonstrate the feasibility of identifying sensitive content, we deployed two IPFS nodes, recording file announcements from nearby peers, and identified over 700 sensitive files.

Furthermore, we deployed honeypot IPFS nodes to gauge potential exploitation of these sensitive files by malicious actors over a six-month period. Our findings indicate that while sensitive files are indeed being shared on the IPFS network, there is currently no evidence of exploitation by attackers. However, with the increasing popularity of IPFS, the risk of such attacks is likely to rise. Our study underscores the importance of acknowledging the risks associated with sharing files on the IPFS network. As IPFS continues to gain traction, proactive measures must be taken to address vulnerabilities and safeguard sensitive data from potential exploitation.

I. Introduction

The InterPlanetary File System (IPFS)[1] is a peer-to-peer distributed storage platform that utilizes decentralized, trustless data storage backed by a distributed hash table. A recent study[2] highlights IPFS's popularity, noting over 3 million web client accesses and billions of daily file shares across more than 300K unique nodes. The platform's broad adoption includes uses from transferring Netflix docker images [3] to native support in browsers like Chromium, Brave, and Opera as an HTTP alternative [4], [5], [6]. IPFS also supports decentralized web applications, social networks, and content search [2].

Given IPFS's decentralized nature, we investigate whether users inadvertently share sensitive files on the platform. Research indicates that file-sharing platforms often host private data inadvertently [7], [8], [9]. On IPFS, shared files are public and retrievable by their Content Identifier (CID), allowing access to anyone with the CID. Even if users remove files, they can persist in the network through caching by intermediate nodes, heightening privacy risks.

To address these issues, we conduct a measurement study and security analysis on sensitive file sharing within IPFS. We define sensitive information as data such as passwords, private keys, and API keys, which if compromised, could enable unauthorized access, financial fraud, or other malicious activities. This study aims to understand the scope of sensitive data exposure and its implications on IPFS.

The main challenge with IPFS is its extensive network size, making it infeasible to search through all files across all peers. We utilize IPFS-search [10], a community-built search engine, similar to Google, which indexes metadata of files without needing their CIDs. This search engine connects to over 40 IPFS nodes worldwide, indexing roughly half a million documents daily. To effectively search for sensitive content within these indexed files, we implemented a series of increasingly complex filters to reduce the occurrence of benign files. This approach includes keyword matching, regular expression matching, and manual inspections to refine the results and remove false positives.

Over a three-month period, we analyzed files using IPFS-Search and identified 10,777 files potentially containing sensitive content. Through our filtering process, we discovered 129 private keys and 23 API keys in uncompressed files, and 1,788 sensitive files in compressed code repositories, mainly containing hardcoded API keys. These keys could enable unauthorized access, posing severe risks to businesses [11].

We also evaluated the security risks from attackers potentially setting up their own IPFS nodes to find sensitive files. Over four months, using two vantage points, we identified 752 sensitive files, demonstrating the limitations of IPFS-Search's safety measures and emphasizing the need for more robust security strategies.

Further, we traced sensitive files back to their original repositories to understand the persistence of sensitive data on IPFS. We found that while 60% of sensitive files in GitHub repositories had been patched, these patched files remained accessible on IPFS. Similarly, 40% of sensitive Node.js files had been patched in their respective online repositories. As part of our responsible disclosure process, we contacted repository owners to help them address these security issues. We continue to update them with our findings to help mitigate risks associated with leaked sensitive information.

Finally, our honeypot experiment aimed to determine if attackers are exploiting sensitive files on IPFS. We deployed 500 decoy files across five locations and monitored them for six months. Our findings showed no attempts by attackers to exploit these files, indicating that the IPFS network has not yet been targeted for such activities.

To conclude, our study reveals the presence of sensitive files being shared on IPFS. These files can be discovered by leveraging centralized search services like IPFS-search using an opportunistic approach. Furthermore, our findings highlight that

ISBN 978-3-903176-63-8 © 2024 IFIP

978-3-903176-63-8/24/$31.00 ©2024 IEEE

the implementation of safety restrictions within these centralized services does not fully mitigate the risk of sensitive file exposure. Attackers can circumvent these restrictions by deploying their own search instances. As the IPFS ecosystem continues to evolve, it becomes crucial to address these vulnerabilities and develop robust security mechanisms to protect against the sharing and discovery of sensitive files.

II. Background and Motivation

A. InterPlanetary File System

In this section, we provide some background on IPFS. The *global IPFS network* is a peer-to-peer network consisting of IPFS nodes. Each IPFS node runs the IPFS protocol and is connected to each other over the WAN. At initialization, each node is assigned a cryptographic hash called *Peer ID*. The IPFS nodes know a subset of all the peers in the global network and will use these peers to search for content.

Unlike traditional URL-based protocols such as HTTP, IPFS uses content-based addressing. IPFS nodes generate a Content Identifier (CID) for each file, which is the hash of the content of the file. The CID along with its provider ID (i.e., the node that has a copy of the content) form a provider record which is stored in a Distributed Hash Table (DHT). This DHT is used to efficiently find the provider record for any CID. At a high level, an IPFS node that wants to retrieve the content of a CID calculates the k closest peers to the CID. The node then asks these peers for the CID's provider record. If the provider record is not located within the k closest peers, each peer in turn contacts their k closest peers and so on until the provider record is found.

B. Motivation: Implications of IPFS File Sharing

At its core, the IPFS platform is used to store and share content *publicly*. This means that a file that is shared over IPFS can be retrieved by anyone if they have the CID. This design choice has an impact on security.

For example, a typical URL like `https://example.com/example.txt` provides straightforward, public access to example.txt. In contrast, a cloud system like OneDrive secures files on its servers, releasing them only through URLs that include secret tokens, like `https://onedrive.live.com/?cid=PGK0TQ6YI0T01AWW&id=PGK0TQ6YI0T01AWW%2198521`. This allows file owners to control access explicitly. However, in the case of IPFS, the URL parameter looks like a secret token, but the parameter is the file's CID. An example would be `https://ipfs.io/ipfs/QmbFMke1KXqnYyBBWxB7L4N4c5SBnJMVAiMNRCGu6x1AwQH`. The CID is announced by IPFS nodes to their closest peers (§ II-A) when the file is added to the network. Unlike OneDrive, anyone with access to the CID can access the content of the file.

The similarity between OneDrive's secret tokens and IPFS's CIDs may cause users to conclude that their files are private and thereby lead them to share private files on the platform, without realizing the implications of that sharing. Similar issues have been observed by other studies on file-sharing platforms [7] where seemingly private-file links can be discovered and accessed by attackers. Further, as discussed earlier, the replication feature of IPFS allows anyone with the CID to download and re-host the content. Combined with the fact that no central authority can force the take-down of content, sensitive files can remain in the network indefinitely, even after their original owners realize their mistake and delete their sensitive files.

In this paper, we characterize the extent of sensitive file sharing on IPFS and discuss its implications in the context of malicious actors opportunistically seeking to identify and gain access to sensitive content.

III. Sensitive File Leaks on IPFS

Our goal is to detect files on IPFS containing sensitive data, defined here as passwords, API keys, cryptographic keys, and crypto-wallets. Studies on GitHub and other repositories use a similar definition, though they typically exclude crypto-wallets. Given IPFS's significant use in blockchain contexts, we include crypto-wallets in our scope [12], [13].

The volume of data shared daily on IPFS is significant, with Protocol Lab reporting over 1 billion new CIDs each day. Consequently, it is impractical to contact all peers and retrieve every file [14], [15].

Our approach utilizes *IPFS-search*, a platform similar to search engines like Google, designed to locate content within the IPFS network. IPFS-search uses custom nodes to detect file shares, retrieve content, and catalog metadata such as file type and title. This metadata is indexed in the IPFS-search database, which processes about 500k documents daily and provides an API for document queries [15].

Although IPFS-search covers only a portion of the IPFS network, our analysis reveals significant sensitive file leaks, demonstrating its effectiveness despite its limited scope.

A. Methodology

Even with the use of IPFS-search, any automated technique that identifies sensitive file leaks can result in a large number of false positives. Manual analysis, while accurate, cannot scale to the vast number of files on IPFS. To address this, we apply progressively complex filters, illustrated in Figure 1. Each filter aims to reduce the number of files potentially containing sensitive content.

Fig. 1: *Different filtering stages to find sensitive files*

a) File-extension and Keyword-based filtering

Using IPFS-search API, we perform queries based on file extensions and keywords that indicate sensitive content, such as *.key, .crt, .ppk,* and *.rsa*, along with phrases like *BEGIN PRIVATE KEY* [12]. We focus this initial filter on metadata rather than the complete content, collecting files from September 21, 2022, to November 30, 2022.

b) Regular expression mapping

We refine our search using regular expression mapping to pinpoint specific content that indicates sensitivity, such as private keys and API keys, expanding on the methods from previous studies [12], [13].

Private Keys: We incorporate regular expressions for SSH and DSA keys into our detection methods as these key types

are frequently leaked in IPFS files [12], [13]. Details of these expressions are provided in Table VI in the Appendix.

API Keys across platforms: We use a set of 13 regular expressions to identify API key leaks, crucial for maintaining service security. The impact of such leaks is significant, as they can enable unauthorized transactions or access to services [12].

c) Removing false positives

To minimize false positives, we compare the hash of each file against all matched files to remove duplicates. Additionally, we manually review files that appear in different formats, such as JSON and YAML, ensuring our final dataset reflects a unique and accurate count of sensitive files.

B. Sensitive leaks in IPFS files: Results

In total, we downloaded 10,777 files of different MIME types that match our extension/keyword-based filtering. Table I classifies the top file categories. Based on the MIME type, the majority of the files are either compressed files (62%) or plain text files (37%). We separate the compressed and uncompressed files and identify sensitive leaks within each set.

TABLE I: *Top 10 MIME Type for 10,777 files downloaded from IPFS-search using the first keyword-based filter*

MIME Type	Count	MIME Type	Count
gzip	4,353	epub+zip	176
plain/text	2,449	octet-stream	84
zip	2,178	html	73
json	902	x-java	35
pgp-signature	396	x-c	34
		other (pgp-keys, python, x-c++ ...)	124

Sensitive file leaks in plain text files

First, we analyze the non-compressed plain text files by applying regular-expression filters. We found 236 file matches with 149 private key matches and 87 matches on API keys. After removing the false positives we identified 129 unique private keys and 23 unique API key matches.

Table II shows the type of private key leaks. 60% of the leaks are RSA private keys, 17% are SSH keys, and the rest of other keys. While the key itself does not contain any host information, the attacker could do further reconnaissance (e.g. from which node was the file retrieved [16]) to identify the specific hosts that could be exploited using these stolen keys. For API key matches, we identified 23 unique API keys belonging to various services, shown in Table III. Most of the credentials are from Google and are all hardcoded into source-code files to interact with Google's services.

Sensitive file leaks in compressed files

We next look at compressed files that match our keyword filters. The files were largely code repositories. Out of the 6,716 compressed files, 61% (4,075) are Node.js libraries from npm, 29% (1,969) are GitHub Go libraries, 6.8% (461) contain some source code in the directory, and the rest are categorized as "other" (i.e., compressed files that are not code repositories).

In total, we identified 9,119 sensitive files based on our regular expression mapping; after removing duplicates, we identified 8,309 unique sensitive files. However, there are still

TABLE II: *A total of 149 private key matches were found using the regular expression matching out of which 129 were unique matches (post false positive removal).*

Private Key Type	Total Match	Unique Match
RSA	82	77
SSH	36	22
General	25	24
EC	4	3
DSA	2	2
PGP	1	1

TABLE III: *A total of 87 API key matches were found using the regular expression matching, and 23 unique key matches were found after removing false positives.*

Platform/API	Total Match	Unique Match
Google OAuthID	25	10
Google API	21	7
Amazon AWS	24	4
Stripe Standard API Key	17	2

false positives since repositories have testing code with valid matches that are not necessarily used in production. For example, the code may contain an example file or a test file with a *dummy* key. To reduce these false positives, we implement an additional filtering strategy where we examine the path of each matched file. If the path or the filename contains one of *test, example, dummy, sample,* or *readme*, we consider them as false positives. After the filtering, we identified 1,788 files with sensitive information.

Fig. 2 shows the distribution of each regular expression match after the filtering. The majority of the sensitive leaks involve Google OAuth IDs (36%) and GoogleAPI keys (18%). The presence of these hardcoded credentials in compressed files matches our earlier observation of hardcoded credentials in non-compressed files. As before, if any of these files fall into the hands of attackers, they can be abused to launch attacks against the owners of the corresponding applications. For instance, previous studies have shown various attacks using OAuth leaks [17], [18], [19], [20].

C. Case study: Leaks in repositories

In our aforementioned analysis, we discovered 4,075 Node.js libraries and 1,969 GitHub Golang library repositories. Sharing libraries over IPFS has additional security implications. Prior work has established that developers inadvertently include sensitive information in code repositories [12]. In the context of this work, if the owner or another user with access to a repository with sensitive information uploads the codebase to IPFS, the sensitive information can persist indefinitely, even after the leak is identified and removed from the original repository. Similarly, if a developer decides to make their repository private, a version of that repository can still be obtained over IPFS.

As a case study, we analyze Node.js and GitHub files shared on IPFS and compare them to their corresponding public repositories. All repositories are publicly available on GitHub, suggesting their intended public nature. In addition, in our responsible disclosure VI, we inform the repository owners about any sensitive leaks

TABLE IV: *Repositories status for Node.js and GitHub Golang libraries shared on IPFS.*

Type	Libraries found on IPFS	Unique Libraries	Corresponding repository found Online
Node.js Library	4,075	1,835	1,727
GitHub Golang Library	1,969	1,043	990

1) Identifying online GitHub/Node repositories

Table IV shows the status of both library types. We identified 1,835 unique Node.js libraries and 1,043 unique GitHub libraries shared on IPFS that have sensitive information (after deleting different versions of the same library). For Github Golang libraries, we tracked 990 of the 1043 libraries to their corresponding

repositories on GitHub online. The remaining 53 were not searchable using the GitHub API. We then queried the owners of these repositories and found that 48 of the owners are still active on GitHub. So we speculate that these repositories were either moved to private repos, or the owners deleted these repositories. For Node.js repositories, for 1,727 of the 1,835 libraries shared on IPFS, we were able to track the original repository.

2) Comparing sensitive leaks on IPFS and the original repository

We next compare each sensitive file shared on IPFS with the original online repository (Node.js or GitHub). The goal here is to characterize the persistence of these sensitive files; in other words, to find if the repository owner removed or patched the sensitive files in the online version. For this study, we focus on warnings regarding Amazon AWS and RSA private keys as they have distinct characteristics.

We categorize the differences as follows:

- **Sensitive information persists**: Sensitive information is present in both the IPFS library and the original repository.
- **Sensitive information patched**: Sensitive information is present in IPFS but does not exist in the original repository (i.e., the file is patched).
- **Sensitive information patched, but new sensitive information present**: Sensitive information is present in IPFS. This sensitive information does not appear in the original repository but the repository has new and different sensitive leaks.
- **Sensitive information removed**: Sensitive information is present in IPFS. But the file that contains sensitive information does not exist in the original repository.

We found for GitHub's repository, nearly 60% of all sensitive information is no longer available. Of these, for over half of them, the original file with sensitive leaks is removed from the GitHub repository's latest version. For the Node.js repository, 70%, of the sensitive information are unchanged. But for information that is changed, the majority of the sensitive information is removed. This difference between GitHub and Node.js is likely because developers are more active on GitHub and fix warnings more frequently.

Further, for all IPFS repositories that contain sensitive information, we compare the time difference between the version that was shared on IPFS and the newest online repository version. We identified that over 50% of the cases, the IPFS version is at least two years older than the current online version. This indicates that the sensitive leaks on IPFS can persist for an extended period of time (as long as at least one node retains a copy of the sensitive file). This also shows that pruning the older versions of code from GitHub to deal with secret leaks is insufficient.

D. Deploying our own monitoring

In the previous section, we used the IPFS-search infrastructure to identify sensitive information leaks. However, IPFS-search is a centralized service and can potentially use filtering or other techniques to stop opportunistic attackers from trivially finding sensitive files.

An alternative technique for attackers is to deploy their own IPFS network monitoring, thereby bypassing any content filtering done by the IPFS-search infrastructure. To characterize the volume of sensitive information attackers could potentially gather using their own IPFS-search instances, we deployed two instances (both on US East Coast) from August 1, 2022, to November 30, 2022. Due to the limitation of the disk space, we only downloaded files that are plain text and compressed files, as they are the most dominant file type from our IPFS-search study (§III-B). In total, we downloaded 1,678,170 files out of 3.1 million unique CIDs collected and most of the files are JSON data files likely because developers use IPFS to share web-related content.

We then applied the methodology described in §III-A. In all, we identified 105 unique files that contain sensitive information. The most frequent leak across non-compressed files are Google API Keys which is similar to our observation when using IPFS-search. Among the 24,331 compressed files, we identified 647 unique files that contain sensitive information. The majority of the sensitive file leaks are Google OAuth ID and Google API keys, again, similar to the study of the sensitive file using IPFS-search.

E. IPFS file availability over time

One of the unique features of IPFS is that files can be cached and replicated throughout the network. On the other hand, a previous study [2] shows that the churn rate of IPFS providers is high. This means providers join and leave the IPFS network relatively quickly, which means that providers may become unavailable quickly. This in-turn affects the availability of files.

To characterize the availability of sensitive files over a long period of time, we conducted the following experiment: for each sensitive file we identified on IPFS, we searched for the file after a 6-month gap. We conducted this experiment for 1,033 CIDs that we identified as sensitive files from both IPFS-search and our own deployment. We found that even after 6 months, over 40% of the files have at least one provider, which means they are still available. More importantly about 20% of the sensitive files have more than one provider; with an extreme case of over 20 providers having a copy of a sensitive file. This result further indicates that once sensitive leaks appear on IPFS, they can persist for an extended time.

F. Takeaway

Our analysis confirms the presence of sensitive files being shared on IPFS, where such sharing can occur accidentally or due to a misunderstanding regarding the public nature of IPFS content.The majority of the identified sensitive files were compressed files associated with code repositories, with API keys being the most commonly leaked information. Exposure of even a single secret from code repositories can have a catastrophic impact on business operations. For instance, the SolarWinds attack, which affected Fortune 500 companies and multiple US government agencies, is believed to have originated from the attacker discovering a weak password within a GitHub repository [21]. This example underscores the importance of safeguarding sensitive information. We also show that, a malicious user can deploy their own IPFS instance with just two vantage points and can identify hundreds of IPFS files that contain sensitive information.

IV. Gauging malicious activity on IPFS

In the previous sections of this paper, we investigated the population of sensitive files on IPFS and highlighted the potential that these files provide to prospective attackers. At the same

time, just because attackers *could* be using the IPFS network to steal sensitive content, does not necessarily mean that they are currently engaging in that activity. To this end, in this section, we report on the findings of deploying our own honeypot experiment involving fake sensitive files (called decoy files) that lead back to monitored infrastructure under our control. Researchers have been using the concept of honeypots for over three decades [22], [23], [24], [7], deploying fake files and fake infrastructure for the express purpose of being compromised, so that attackers can be studied while keeping them away from real production systems.

A. Setup

To set up the honeypot experiment, we craft different types of decoy files that will be uploaded to the IPFS network:

- **HTML:** the file contains a redirect link (also known as a "beacon") which will notify us that the file was opened.
- **Microsoft Word, PDF:** the file contains login credentials to our honeypot server. In addition, the file embeds a beacon that will trigger upon opening the document.
- **SSH Private Keys:** the private key can be used to directly login to our honeypot server.
- **Cryptocurrency wallet:** contains seed/private-key data allowing attackers to steal a small amount of funds.
- **Control files:** randomly generated files.

To further attract attackers, all the files are given attractive naming such as online logins, password backups, etc.

B. Data generation

To simulate accidental leaks we utilized an online fake information generator [1] to generate 300 unique people with usernames, passwords, as well as fake banking information. Further, we registered 5 domain names and point them to our honeypot server so that each password login leak corresponds to one of these domain names.

C. Deployment

To upload the decoy files to the IPFS network, we deployed 5 IPFS nodes in the US, UK, Brazil, Japan, and Australia and uploaded 20 HTML, PDF, Microsoft Word, and SSH Private keys from each node. We also uploaded 30 control files and one wallet file. Since IPFS uses content addressing, when a single file is uploaded to the network, the file name will not be retained. To address this downside (downside in terms of discoverability by prospective attackers), we utilized the so-called *wrapped* option [4] which will wrap the file into a single directory and thus retain the filename when uploaded. The honeypot server was deployed in the US.

Each IPFS node re-announces the provider record to its peers every 12 hours. Further, for every 12 hours each IPFS node will retrieve all the decoy files from the other four so that the CID will spread across the network.

[1]https://www.fakenamegenerator.com/

[4]https://docs.ipfs.tech/reference/kubo/cli/#ipfs-add

D. Results

We deployed our honeypot from August 5, 2022, to February 16, 2023. In total, we observed that 56 decoy files were downloaded across various categories. Figure 3 shows the downloaded file-type distribution with respect to each IPFS node, where no clear download pattern emerges. While our decoy files were downloaded by a number of IPFS peers, we did not observe any malicious action against our honeypot server. To understand the characteristics of the peers who downloaded our files, we used their IP addresses to obtain geolocation and ASN information. The majority of these clients were located in Germany (shown in Figure 4) and the ASes belong to *Hetzner Online GmbH*. We confirm that these peers are IPFS-search nodes using a reverse DNS lookup.

E. Takeaway

At the time of our honeypot experiments, we observed only machine-to-machine traffic to our IPFS nodes, and the majority of our files were indexed by IPFS-search. While this is good news for the owners who leak files that contain sensitive content on IPFS, however, these files can stay on IPFS indefinitely, essentially waiting for future attackers to discover them.

V. Related Work

Studies on IPFS have mainly focused on evaluating IPFS performance [2], network size [25], the transport layer protocol Bitswap [26], and I/O performance [27]. There are fewer studies on the security implications of IPFS. One study identifies critical security issues regarding Sybil and Eclipse attacks on the IPFS network [28]. The study found that a single attacker can manipulate the network by generating massive peer IDs and flooding the network. Once the network is saturated with fake peers, then the attacker can advertise fake routing information to the victim's IPFS nodes and isolate the victim from the network. Another study focused on how IPFS is being utilized by ransomware services [29] where attackers host Web pages that ask for ransom on IPFS, benefiting from the robustness and resilience of IPFS. Related research has also discovered that botnets deployed into the IPFS networks enable attackers to exert fine-grained control over their victims [30].

To the best of our knowledge, while studies exist for characterizing the presence of sensitive content on centralized platforms (such as Github [12] and file-hosting services [7]), there has been no study of sensitive-file sharing on IPFS. The core functionality of IPFS and its intended use for building decentralized applications makes it different from GitHub and therefore meriting its own investigation.

VI. Ethics

Even though all the files that we accessed in this study are by definition public, the entire premise of this paper is that some of the files stored on IPFS are sensitive in nature. As such, in this section, we describe how we conducted our experiments to ensure that we preserve the privacy of users and the overall ethics of our work.

Our focus is solely on identifying whether sensitive files are being shared on IPFS, without revealing the providers (i.e. potential users) of these files. Throughout our data collection process via IPFS-search and our own search instance, we retrieved files without recording any information about the

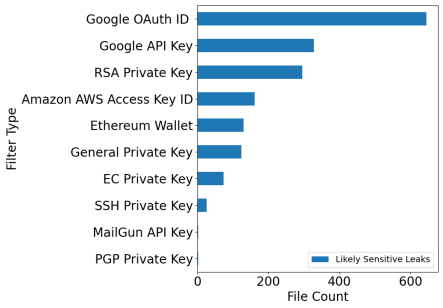

Fig. 2: Among the compressed gzip and zip files, 1788 contained sensitive information

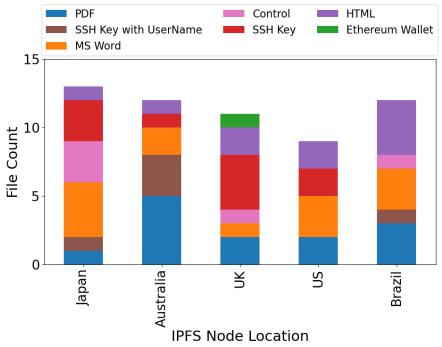

Fig. 3: Downloaded decoy files distribution for each deployed IPFS node. Not every file is retrieved by other peers.

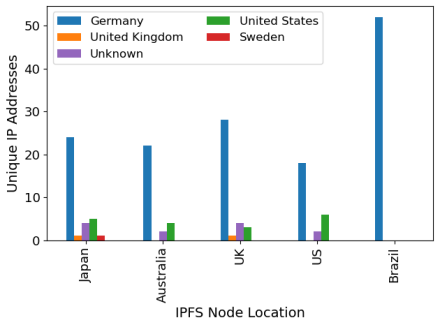

Fig. 4: All IP Geo-location for peers downloaded decoy files with respect to each deployed IPFS node.

providers, ensuring their privacy. We also took measures to prevent the unintentional spreading of these files by configuring our IPFS nodes to refrain from caching and redistributing them. This approach minimized our impact on the ecosystem and avoided further propagation of potentially sensitive files.

Moreover, due to ethical considerations and the vast volume of collected files, we relied on pattern detection using regular expressions to identify sensitive files. This approach reduced the need for manual analysis of sensitive files outside of the ones containing API keys, encryption keys, etc. Additionally, we removed all collected files post-analysis to maintain data integrity and privacy.

Finally, responsible disclosure was another key aspect of our study. We notified repository owners about sensitive leaks within code repositories via email and offered assistance to address these issues. We contacted 59 individual developers for Golang libraries. However, due to limitations with the Node.js repository, we were unable to contact its owners. This responsible disclosure provides developers with an opportunity to patch vulnerabilities, such as obtaining new API keys, thereby enhancing the overall security and privacy of the IPFS file ecosystem. We will update the paper with any responses from developers.

VII. Discussion and Limitation

First and foremost, files indexed by IPFS-search and files found using our own deployment are only a fraction of all the files available on IPFS. As such, this work does not measure an upper bound on sensitive leaks on IPFS. Interestingly, even with the limited vantage points that we deployed, we were able to reveal that considerable sensitive leaks do occur on IPFS. New research is needed to explore the *extent* of sensitive file leaks on IPFS. We will release our measurement and analysis code for other researchers to build on.

Additionally, the regular expressions we used to identify sensitive files were restricted to well-defined domains. However, sensitive leaks can vary from person to person such as passwords, phone numbers, or personal identification information. For example, during our manual inspection, we found some configuration files containing credentials from one-off platforms, such as database credentials and cryptocurrency-exchange API keys. It is difficult to identify these leaks using regular expressions because of the number of regular expressions that will be needed to cover these one-off security keys. Expanding the study to search for these additional credentials will further increase the number of sensitive files we can uncover. Similarly, our honeypot experiment focused on a limited set of well-defined sensitive files over a brief observation period; a broader range of file types and a longer duration could yield further discoveries.

Finally, we address potential solutions for enhancing file-sharing privacy on the IPFS network. One such proposal, suggested by Protocol Lab, the main contributor to IPFS, is the DHT Reader Privacy Upgrade [31]. This initiative aims to improve privacy by encrypting CID requests during file announcement and retrieval processes. Currently, when a user requests CID content, the IPFS node forwards the request to nearby peers, potentially revealing the requested CID. With encryption, however, the CID remains concealed, preventing peers from recording it. This enhancement is expected to bolster the anonymity of sensitive files by limiting the exposure of CIDs to peers. It's important to note that this idea is still in the proposal phase and has not been implemented or validated. In our future research, we plan to assess the effectiveness of the DHT Reader Privacy Upgrade, evaluating both its performance and its ability to protect privacy.

VIII. Conclusions

InterPlanetary File System (IPFS) is a peer-to-peer hypermedia sharing protocol with the stated goal of making the web faster, safer, and more open. With its rising popularity and openness, our work aims to answer the following question: *Are users inadvertently sharing sensitive files on IPFS?*

To answer this question, we conducted a measurement study to identify sensitive file leaks. We use IPFS-search, a community-built search engine, as our vantage point, and show that there are thousands of IPFS files that are publicly accessible that share sensitive information including private cryptographic keys and API tokens. Even if a platform like IPFS-search restricts users from searching for sensitive files, we show that, a malicious user can deploy their own IPFS vantage points to search for sensitive files on IPFS. With only 2 vantage points and over a 4 month period, we identified hundreds of sensitive files. Finally, to investigate whether sensitive files are being actively exploited by attackers, we deployed honeypot IPFS nodes that upload decoy files onto the IPFS network. Even though we observed decoy files being downloaded by other peers, no malicious actions were performed using the sensitive content. Our work concludes that sensitive files

are in fact currently shared through IPFS, even though they are not being weaponized by bad actors yet. Given the growing popularity of IPFS and decentralized platforms, our study shows that public sharing of sensitive content on these platforms warrants the attention and additional research by the community, in order to devise methods that protect users while not compromising on the decentralized nature of the underlying protocols.

IX. Acknowledgements

We thank all the reviewers for their constructive feedback. This work was supported in part by NSF grants CNS-1941617, CNS-2211575, and CNS-1909356.

References

[1] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.
[2] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, and Y. Psaras, "Design and evaluation of ipfs: a storage layer for the decentralized web," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 739–752.
[3] E. L. Dirk McCormick, "New improvements to ipfs bitswap for faster container image distribution," https://blog.ipfs.io/2020-02-14-improved-bitswap-for-container-distribution/, 2020.
[4] "Adding ipfs protocol support to chromium," Nov 2022. [Online]. Available: https://blog.ipfs.tech/14-11-2022-igalia-chromium/
[5] B. Bondy, "Ipfs support in brave," https://brave.com/ipfs-support/, 2021.
[6] S. Batt, "Your files for keeps forever with ipfs," https://blogs.opera.com/tips-and-tricks/2021/02/opera-crypto-files-for-keeps-ipfs-unstoppable-domains/, 2021.
[7] N. Nikiforakis, M. Balduzzi, S. Van Acker, W. Joosen, and D. Balzarotti, "Exposing the lack of privacy in file hosting services," *LEET*, 2011.
[8] B. Kaleli, M. Egele, and G. Stringhini, "On the perils of leaking referrers in online collaboration services," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, R. Perdisci, C. Maurice, G. Giacinto, and M. Almgren, Eds. Cham: Springer International Publishing, 2019, pp. 67–85.
[9] M. E. Johnson, D. McGuire, and N. D. Willey, "The evolution of the peer-to-peer file sharing industry and the security risks for users," in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, 2008, pp. 383–383.
[10] "Ipfs search." [Online]. Available: https://ipfs-search.com
[11] D. Goodin, "Thousands of servers found leaking 750mb worth of passwords and keys," Mar 2018. [Online]. Available: https://arstechnica.com/information-technology/2018/03/thousands-of-servers-found-leaking-750-mb-worth-of-passwords-and-keys/
[12] M. Meli, M. R. McNiece, and B. Reaves, "How bad can it git? characterizing secret leakage in public github repositories." in *NDSS*, 2019.
[13] V. S. Sinha, D. Saha, P. Dhoolia, R. Padhye, and S. Mani, "Detecting and mitigating secret-key leaks in source code repositories," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 396–400.
[14] D. Trautwein, "Network-measurements/results/rfm21-hydras-performance-contribution.md at master · protocol/network-measurements," Jan 2023. [Online]. Available: https://github.com/protocol/network-measurements/blob/master/results/rfm21-hydras-performance-contribution.md
[15] S. Henningsen, M. Florian, S. Rust, and B. Scheuermann, "Mapping the interplanetary filesystem," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 289–297.
[16] L. Balduf, S. Henningsen, M. Florian, S. Rust, and B. Scheuermann, "Monitoring data requests in decentralized data storage systems: A case study of ipfs," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022, pp. 658–668.
[17] S. Farooqi, F. Zaffar, N. Leontiadis, and Z. Shafiq, "Measuring and mitigating oauth access token abuse by collusion networks," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 355–368. [Online]. Available: https://doi.org/10.1145/3131365.3131404
[18] D. Fett, R. Küsters, and G. Schmitz, "A comprehensive formal security analysis of oauth 2.0," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1204–1215. [Online]. Available: https://doi.org/10.1145/2976749.2978385
[19] S.-T. Sun and K. Beznosov, "The devil is in the (implementation) details: An empirical analysis of oauth sso systems," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 378–390. [Online]. Available: https://doi.org/10.1145/2382196.2382238
[20] T. Lodderstedt, M. McGloin, and P. Hunt, *OAuth 2.0 Threat Model and Security Considerations*, Jan 2013, no. RFC 6819. [Online]. Available: https://datatracker.ietf.org/doc/rfc6819/
[21] G. Sands, Brian Fung, "Former solarwinds ceo blames intern for "solarwinds123" password leak — cnn politics," Feb 2021. [Online]. Available: https://www.cnn.com/2021/02/26/politics/solarwinds123-password-intern/index.html
[22] C. Stoll, *The cuckoo's egg: tracking a spy through the maze of computer espionage*. Simon and Schuster, 1990.
[23] N. Provos and T. Holz, *Virtual honeypots: from botnet tracking to intrusion detection*. Pearson Education, 2007.
[24] B. M. Bowen, S. Hershkop, A. D. Keromytis, and S. J. Stolfo, "Baiting inside attackers using decoy documents," in *Security and Privacy in Communication Networks: 5th International ICST Conference, SecureComm*. Springer, 2009, pp. 51–70.
[25] E. Daniel and F. Tschorsch, "Passively measuring ipfs churn and network size," in *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2022, pp. 60–65.
[26] A. De la Rocha, D. Dias, and Y. Psaras, "Accelerating content routing with bitswap: A multi-path file transfer protocol in ipfs and filecoin," 2021.
[27] J. Shen, Y. Li, Y. Zhou, and X. Wang, "Understanding i/o performance of ipfs storage: a client's perspective," in *Proceedings of the International Symposium on Quality of Service*, 2019, pp. 1–10.
[28] B. Prünster, A. Marsalek, and T. Zefferer, "Total eclipse of the heart–disrupting the {InterPlanetary} file system," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3735–3752.
[29] C. Karapapas, I. Pittaras, N. Fotiou, and G. C. Polyzos, "Ransomware as a service using smart contracts and ipfs," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2020, pp. 1–5.
[30] A. T. Research, "The interplanetary storm: New malware in wild using interplanetary file system's (ipfs) p2p network," https://www.anomali.com/blog/the-interplanetary-storm-new-malware-in-wild-using-interplanetary-file-systems-ipfs-p2p-network, 2019.
[31] [Online]. Available: https://github.com/ipfs/specs/pull/373

Appendix

A. Regular Expressions

TABLE V: *Regular expressions targeting different service platforms' API Key*

Platform/API	Key Type	Target Regular Expression
Amazon AWS	Access Key ID	AKIA[0-9A-Z]{16}
Amazon MWS	Auth Token	amzn\.mws\.[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}
Google	API Key	AIza[0-9A-Za-z\-_]{35}
	OAuth ID	[0-9]+-[0-9A-Za-z_]{32}\.apps\.googleusercontent\.com
Stripe	Standard API Key	sk_live_[0-9a-zA-Z]{24}
	Restricted API Key	rk_live_[0-9a-zA-Z]{24}
Square	Access Token	sq0atp-[0-9A-Za-z\-_]{22}
	OAuth Secret	sq0csp-[0-9A-Za-z\-_]{43}
PayPal Braintree	Access Token	access_token$production$[0-9a-z]{16}$[0-9a-f]{32}
Meta	Access Token	EAACEdEose0cBA[0-9A-Za-z]+
Twilio	API Key	SK[0-9a-fA-F]{32}
MailGun	API Key	key-[0-9a-zA-Z]{32}
Picatic	API Key	sk_live_[0-9a-z]{32}

TABLE VI: *Regular expression to identify private keys and they have a distinct structure mainly due to their PEM header*

Asymmetric Key Type	Target Regular Expression	Asymmetric Key Type	Target Regular Expression
RSA Private Key	—BEGIN RSA PRIVATE KEY— [\r\n]+(?:\w+:.+)*[\s]* (?:[0-9a-zA-Z+\/=]{64,76}[\r\n]+)+ [0-9a-zA-Z+\/=]+[\r\n]+ —END RSA PRIVATE KEY—	PGP Private Key	—BEGIN PGP PRIVATE KEY BLOCK— [\r\n]+(?:\w+:.+)*[\s]* (?:[0-9a-zA-Z+\/=]{64,76}[\r\n]+)+ [0-9a-zA-Z+\/=]+[\r\n]+= [0-9a-zA-Z+\/=]{4}[\r\n]+ —END PGP PRIVATE KEY BLOCK—
EC Private Key	—BEGIN EC PRIVATE KEY— [\r\n]+(?:\w+:.+)*[\s]* (?:[0-9a-zA-Z+\/=]{64,76}[\r\n]+)+ [0-9a-zA-Z+\/=]+[\r\n]+ —END EC PRIVATE KEY—	SSH Private Key	—BEGIN OPENSSH PRIVATE KEY— [\r\n]+(?:\w+:.+)*[\s]* (?:[0-9a-zA-Z+\/=]{64,76}[\r\n]+)+ [0-9a-zA-Z+\/=]+[\r\n]+ —END OPENSSH PRIVATE KEY—
DSA Private Key	—BEGIN DSA PRIVATE KEY— [\r\n]+(?:\w+:.+)*[\s]* (?:[0-9a-zA-Z+\/=]{64,76}[\r\n]+)+ [0-9a-zA-Z+\/=]+[\r\n]+ —END DSA PRIVATE KEY—	General Private Key	—BEGIN PRIVATE KEY— [\r\n]+(?:\w+:.+)*[\s]* (?:[0-9a-zA-Z+\/=]{64,76}[\r\n]+)+ [0-9a-zA-Z+\/=]+[\r\n]+ —END PRIVATE KEY—