Exposing the Lack of Privacy in File Hosting Services

Nick Nikiforakis[1], Marco Balduzzi[2], Steven Van Acker[1], Wouter Joosen[1], and Davide Balzarotti[2]

[1]DistriNet, Katholieke Universiteit Leuven, Belgium
[2]Institute Eurecom, Sophia Antipolis, France

Abstract

File hosting services (FHSs) are used daily by thousands of people as a way of storing and sharing files. These services normally rely on a security-through-obscurity approach to enforce access control: For each uploaded file, the user is given a secret URI that she can share with other users of her choice.

In this paper, we present a study of 100 file hosting services and we show that a significant percentage of them generate secret URIs in a predictable fashion, allowing attackers to enumerate their services and access their file list. Our experiments demonstrate how an attacker can access hundreds of thousands of files in a short period of time, and how this poses a very big risk for the privacy of FHS users. Using a novel approach, we also demonstrate that attackers are aware of these vulnerabilities and are already exploiting them to get access to other users' files. Finally we present *SecureFS*, a client-side protection mechanism which can protect a user's files when uploaded to insecure FHSs, even if the files end up in the possession of attackers.

1 Introduction

In an ever expanding Internet, an increasing number of people utilize online services to share digital content. Most of the platforms that support file sharing operate in a broadcasting way. For example, users of social networks can upload multimedia files which are then accessible to their entire friend list. In traditional peer-to-peer networks (e.g., DC++, KaZaa and BitTorrent), users share their files with everyone connected to the same network.

Sometimes however, users may want to share files only with a limited number of people, such as their co-workers, spouses, or family members. In these situations, the all-or-nothing approach of sharing files is not desired. While emailing digital content is still a popular choice, most email providers do not usually allow attachments that exceed a few tens of megabytes.

One of the first services designed to fill this gap was offered by RapidShare[1], a company founded in 2006. RapidShare provides users the ability to upload large files to their servers and then share the links to those files with other users. RapidShare's success spawned hundreds of file hosting services that compete against each other for a share of users. Apart from being used as a way to share private files, researchers have found that FHSs are also used as an alternative to peer-to-peer networks [1], since they offer several advantages such as harder detection of the user who first uploaded a specific file, always-available downloads and no upload/download ratio [2] measurements.

In this paper, we present our study on 100 file hosting services. These services adopt a security-through-obscurity mechanism where a user can access the uploaded files only by knowing their correct download URIs. While these services claim that these URIs are secret and cannot be guessed, our study shows that this is far from being true. A significant percentage of FHSs generate the "secret" URIs in a predictable fashion, allowing attackers to easily enumerate their files and get access to content that was uploaded by other users.

We implemented crawlers for different hosting providers and, using search engines as a privacy classification mechanism, we were able to disclose hundreds of thousands of private files in less than a month. The second contribution of this paper is a technique to measure whether attackers are already abusing FHSs to access private information. To reach this goal, we created a number of "HoneyFiles", i.e. fake documents that promise illegal content, and we uploaded them to many FHSs. These files were designed to silently contact one of our servers once they were opened. Over the period of a month, more than 270 file accesses from over 80 different IP addresses were recorded, showing that private documents uploaded to file hosting services are in fact

[1]RapidShare AG, `http://www.rapidshare.com/`
[2]A user-quality metric used mainly in private BitTorrent trackers

actively downloaded by attackers.

The rest of our paper is structured as follows. In Section 2 we present a high-level view of how FHSs operate, followed by a security and privacy study of the top 100 FHSs in Section 3. Section 4 provides the details and reports the results of our HoneyFiles experiment. In Section 5 we present a possible countermeasure to protect users of vulnerable FHSs. We then present ethical considerations regarding our experiments in Section 6. Section 7 explores related work, Section 8 discusses possible future work, and finally, Section 9 concludes the paper.

2 Life cycle of files on File Hosting Services

In this section, we describe the typical life cycle of a file in relation to file hosting services and we point out the privacy-sensitive steps. In order to model this life cycle we consider a FHS which allows users to upload files without the need to register an account.

A typical user interaction with a FHS usually follows the following steps:

1. *Alice* decides to share some digital content[3] with other users by uploading it to her favorite FHS.
2. The FHS receives and stores the file. It then creates a *unique identifier* and binds it to the uploaded file.
3. The identifier (ID) is returned to *Alice* in the form of a *URI* that permits her to easily retrieve the uploaded file.
4. *Alice* may now distribute the URI according to the nature of her file: If the file is meant to be public, she may post the link on publicly accessible forums, news-letters or social-networks, otherwise she may prefer to share the URI using one-to-one communication channels such as e-mails or instant messaging.
5. The public or private recipients use the shared URI to access the file that *Alice* has uploaded.
6. If the uploaded file violates copyright laws (movies, music, non-free software) then a third party can possibly report it. In this case the file is deleted and the unique ID is possibly reused. If the file survives this process, it remains accessible for a limited amount of time (set by the provider) or until *Alice* voluntarily removes it.

In the context of this paper, we focus on the scenario in which the user uploads a *private* file to a FHS and uses a one-to-one channel to share the unique ID returned by the service with the intended recipients of the file. We also assume that the trusted recipients will not forward the URI to other untrusted users.

In this case, the only way in which the file can be accessed by a malicious user is either by guessing the unique ID returned by the FHS or by exploiting a software bug that will eventually allow access to the uploaded files.

[3] In the rest of the paper called "file"

3 Privacy study

The first phase of our study consists in comparing the privacy offered by 100 File Hosting Services. The list was constructed by merging the information retrieved from different sources, such as the Alexa website, Google search engine, and a number of articles and reviews posted on the Web [9, 14].

Our final top 100 list includes well-known FHSs like `RapidShare`, `FileFactory` and `Easyshare` as well as less popular and regional websites like `FileSave.me` (India), `OnlineDisk.ru` (Russia) and `FileXoom.com` (Mexico).

Before starting the experiments we manually screened each website and found that 12 of them provide either a search functionality (to retrieve the link of a file knowing its name) or an online catalogue (to browse part of the uploaded files). Obviously, these services are not intended for storing personal documents, but only as an easy way to publicly share files between users. Therefore, we removed these FHSs from our privacy study and we focused our testing on the remaining 88 services.

We began our evaluation by analyzing how the unique *file identifiers* are generated by each FHS. As we described in the previous section, when a user uploads a file to a FHS, the server creates an unique identifier (ID) and binds it to the uploaded file. The identifier acts as a shared secret between the user and the hosting service, and therefore, it should not be possible for an attacker to either guess or enumerate valid IDs.

Sequential Identifiers

For each FHS, we consecutively uploaded a number of random files and we monitored the download URIs generated by the hosting service. Surprisingly, we noticed that 34 out of the 88 FHSs (38.6%) generated sequential IDs to identify the uploaded files. Hence, a hypothetical attacker could easily enumerate all private files hosted by a vulnerable FHS by repeatedly decreasing a valid ID (that can be easily obtained by uploading a test file).

As an example, the following snippet shows how one of the hosting providers (anonymized to *vulnerable.com*) stored our uploaded files with sequential identifiers:

```
http://vulnerable.com/9996/
http://vulnerable.com/9997/
http://vulnerable.com/9998/
http://vulnerable.com/9999/
[...]
```

However, the enumeration attack is possible only when the URI does not contain other *non-guessable* parameters. In particular, we noticed that out of the 34 FHS

	Sequential ID	Non-Sequential ID	Tot
Filename:			
Required	14	6	20
Not required	20	48	68
Total	34	54	88

Table 1: Analysis of the Download URI's identifier

that use sequential identifiers, 14 also required the proper filename to be provided with the secret identifier. For example, the URI was sometimes constructed by appending the filename as shown in the following example:

```
http://site-one.com/9996/foo.pdf
http://site-one.com/9997/bar.xls
http://site-one.com/9998/password.txt
[...]
```

Since the filenames associated to each ID are, in general, unknown to the attacker, this feature acts as an effective mitigation against enumeration attacks. Even though it would still be possible for an attacker to narrow down his research to a particular filename, we did not investigate this type of targeted attacks in our study.

Table 1 provides an overview of the techniques adopted by the various FHSs to generate the download URIs. The privacy provided by 20 service providers was extremely weak, relying only on a sequential ID to protect the users' uploaded data. Unfortunately, the problem is extremely serious since the list of insecure FHSs using sequential IDs also includes some of the most popular names, often highly ranked by Alexa in the list of the top Internet websites.

To further prove that our concerns have a practical security impact, we implemented an automatic enumerator for the 20 FHSs that use sequential IDs and do not require the knowledge of the associated filename. Our tool inserted a random delay after each request to reduce the impact on the performance of the file hosting providers. As a result, our crawler requests were interleaved with many legitimate user requests, a fact which allowed us to conduct our experiment for over a month without being blacklisted by any service provider.

When a user requests a file by providing a valid URI, the FHS usually returns a page containing some information about the document (e.g., filename, size, and number of times it was downloaded), followed by a series of links which a user must follow to download the real file. This feature is extremely convenient for an attacker that can initially scrape the name of each file, and then download only those files that look more interesting.

By enumerating sequential IDs, our crawler was able to retrieve information about 310,735 unique files in a period of 30 days. While this list is "per se" sensitive information, we tried to estimate how many files correspond to private users' documents.

It is reasonable to assume that if the user wants to make her file publicly accessible, she would post the download URI on websites, blogs, and/or public forums, depending on the target audience of the file. On the other hand, if a file is intended to be kept private, the link would probably be shared in a one-to-one fashion, e.g., through personal emails and instant messaging. We decided to exploit this difference to roughly characterize a file as *public* or *private*. In particular, we queried for each file name on Bing, Microsoft's search engine, and we flagged as "public" any file whose link was found on the Web. If Bing did not return any results for our query, we considered the file as private.

File Type	# Enumerated Files
Images (JPG, GIF, BMP)	27,771
Archives (ZIP)	13,354
Portable Document Format (PDF)	7,137
MS Office Word Documents	3,686
MS Office Excel Sheets	1,182
MS PowerPoint	967

Table 2: Number of files with sensitive types reported as private by our privacy-classification mechanism

Out of the 310,735 unique filenames extracted with our enumeration tool, Bing returned no search results for 168,320, thus classifying 54.16% of files as private. This classification is quite approximate as the list of private files also contains data exchanged in closed "pirate" communities, beyond the reach of search engines' crawlers [1, 8]. Nevertheless, this technique still provides a useful estimation of the impact of this kind of attack on the privacy of FHS users.

Table 2 shows the number of unique private files crawled by our tool, grouped by common filetypes. In addition to the major extensions reported in the Table, we also found several files ending with a `.sql` extension. These files are probably database dumps that attackers may use to gain a detailed view of the content of the victim's database.

Even though we believe that the majority of these files contain private information, for ethical reasons we decided not to verify our hypothesis. In fact, to preserve the users' privacy, our crawler was designed to extract only the name and size of the files from the information page, without downloading the actual content.

Random Identifiers

In a second experiment we focused on the 54 FHSs that adopt *non sequential* identifiers (ref. Table 1).

In these cases, the identifiers were randomly generated for each uploaded file, forcing a malicious user to guess a valid random value to get access to a single file. The complexity of this operation depends on the length (i.e., number of characters) of the secret and on the character set (i.e., number of possible values for each character) that is used to generate the identifier. As shown in Figure 1 and Figure 2, different FHSs use different tech-

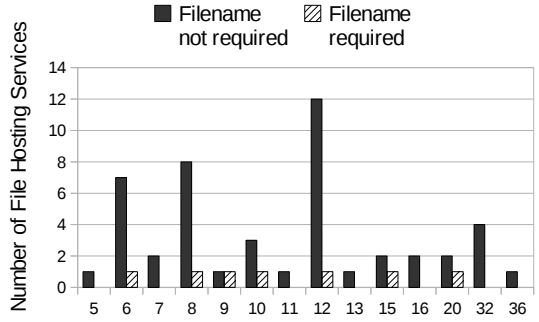

Figure 1: Length of the Identifier

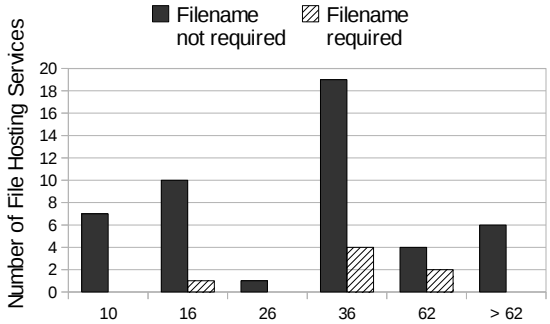

Figure 2: Size of the Identifier's Character Set

Length	Chars Set	# Tries	Files Found
6	Numeric	617,169	728
6	Alphanumeric	526,650	586
8	Numeric	920,631	332

Table 3: Experiment on the non-sequential identifiers

niques, varying in length from 6 to 36 bytes and adopting different character sets.

The three peaks in Figure 1 correspond to identifier length of six, eight, and twelve characters respectively. The second graph shows instead that the most common approach consists in using alphanumeric characters. Unfortunately, a fairly large number of FHSs are still adopting easily guessable identifiers composed only of decimal or hexadecimal digits.

To show the limited security offered by some of the existing solutions we conducted another simple experiment. We modified our enumerator tool to bruteforce the file identifiers of three different *non sequential* FHSs that did not require the filename in the URI. In particular, we selected a FHS with numeric IDs of 6 digits, one with numeric IDs of 8 digits, and one with alphanumeric IDs of 6 characters. Our tool ran for five days, from a single machine on a single IP address. The results, shown in Table 3, confirm that if the random identifiers are too weak, an attacker can easily gain access to thousands of third-party files in a reasonable amount of time.

It is also interesting to note that the success rate of guessing both numeric and alphanumeric IDs of six digits was about the same (1.1 hit every thousand attempts).

Feature	Number of FHSs
CAPTCHA	30%
Delay	49%
Password	26%
PRO Version	53%
Automated File Trashing (2-360 days)	54%

Table 4: Security features

This shows that the size of the space to explore is not the only variable in the process and that the important factor is the ratio between the number of possible identifiers and the total number of uploaded files.

Existing Mitigations

We have shown that different hosting providers, by adopting download URIs that are either sequential or easily predictable, can be abused by malicious users to access a large amount of private user data. However, some hosting providers implement a number of mechanisms that mitigate the possible attacks and make automated enumeration more difficult to realize.

From a manual analysis of the FHSs in our list we identified two techniques commonly used to prevent automated downloads. As summarized in Table 4, 30% of websites use CAPTCHA and 49% force the user to wait (between 10 to 60 seconds) before the download starts. However, note that both techniques only increase the difficulty of downloading files, and not the process of guessing the right identifier. As a consequence, our tool was not restricted in any way by these protection mechanisms. In addition, we noticed that a large number of sites offer a *PRO version* of their service where these "limitations" are removed by paying a small monthly fee.

A much safer solution consists in adding an online password to protect a file. Unfortunately, as shown in Table 4, only a small percentage of the tested FHSs provided this functionality.

Other Design and Implementation Errors

According to the results of our experiments, many FHSs are either vulnerable to sequential enumeration or they generate short identifiers that can be easily guessed by an attacker. In the rest of this section, we show that, even when the identifiers are strong enough to resist a direct attack, other weaknesses or vulnerabilities in the FHS's software may allow an attacker to access or manipulate private files.

While performing our study, we noticed that 13% of hosting services use the same, publicly available, software to provide their services. To verify our concern about the security of these systems, we downloaded and audited the free version of that platform. Through a manual investigation we were able to discover serious design and implementation errors. For instance, the software contained a directory traversal vulnerability that allows

an attacker to list the URIs of all the recently uploaded files.

In addition, the software provides to the user a delete URI for each uploaded file. The delete URI can be used by the user at any time to delete her uploaded files from the service's database. This feature can improve the user's privacy since the file can be deleted when it is no longer needed. The deletion ID generated by this software was 14 characters long, with hexadecimal characters. This provides 16^{14} valid combinations which make any attack practically impossible. However, we noticed that the "report file" link, a per file automatically generated link to report copyright violations, consisted of the first 10 characters of the deletion code.

Since the "report file" link (used to report a copyright violation) is publicly available to everybody, an attacker can use it to retrieve the first 10 digits of the delete URI, thus lowering the number of combinations to bruteforce to only $16^4 = 65,536$.

To conclude, it is evident that even if end-users only share the download-link with their intended audience, they can not be certain that their files will not reach unintended recipients. In Section 5, we will discuss a client-side solution that can protect a user's files without changing his file-uploading and file-sharing habits.

4 HoneyFiles

In Section 3 we showed that a significant percentage of file hosting services use a URI generation algorithm that produces sequential identifiers, allowing a potential attacker to enumerate all the files uploaded by other users. In addition, our experiments also showed that some of the FHSs which use a more secure random algorithm, often rely on weak identifiers that can easily be bruteforced in a short amount of time.

In summary, a large amount of the top 100 file hosting services are not able to guarantee to the user the privacy of her documents. The next question we investigate is whether (and to what extent) the lack of security of these websites is already exploited by malicious users. In order to answer this question we designed a new experiment inspired by the work of Bowen et al. [3] and Yuill et al. [18] on the use of decoy documents to identify insider threats and detect unauthorized access.

First, we registered the *card3rz.co.cc* domain and created a fake login page to a supposedly exclusive underground "carding"[4] community. Second, we created a number of decoy documents that promised illegal/stolen data to the people that accessed them. Each file contained a set of fake sensitive data and some text to make the data convincing when necessary. The first two columns of Table 5 list the names and descriptions of each file. The most important characteristic of these files is the fact that, once they are open, they automatically connect back to our monitor running on the *card3rz.co.cc* server. This was implemented in different ways, depending on the type of the document. For example, the HTML files included an `<img/>` tag to fetch some content from our webpage, the `exe` file opened a TCP connection upon its execution, and the PDF documents asked the user permission to open a webpage. For Microsoft's DOC format, the most straightforward way we found was to embed an HTML file inside the document. In cases where user action was required (e.g., the acceptance of a warning dialog or the double-click of the HTML object in the DOC file) we employed social engineering to convince the malicious user to authorize the action.

In addition to the connect-back functionality, one of the files contained valid credentials for logging into our fake carding website. We did this to investigate whether attackers would not only access illegally obtained data but also take action on the information found inside the downloaded files.

The last step of our experiment consisted in writing a number of tools to automatically upload the HoneyFiles to the various FHSs. Since the links to our files were not shared with anyone, any file access recorded by our monitor was the consequence of a malicious user that was able to download and open our HoneyFiles, thus triggering the hidden connect-back functionality.

Monitoring Sequential FHSs

In our first experiment we used our tools to upload the HoneyFiles 4 times a day to all the FHSs adopting sequential identifiers. We also included the ones that have Search/Catalogue functionality in order to find out whether attackers search for illegal content in FHSs.

While we were initially skeptical of whether our experiment would provide positive results, the activity recorded on our monitor quickly proved us wrong. Over the span of one month, users from over 80 unique IP addresses accessed the HoneyFiles we uploaded on 7 different FHSs for a total of 275 times. Table 6 shows the categorization of the attackers by their country of origin using geolocation on their IP addresses. While most of the attacks originated from Russia, we also recorded accesses from 16 other countries from Europe, the United States and the Middle East, showing that this attack technique is used globally and it is not confined to a small group of attackers in a single location.

The third column of Table 5 presents the download ratio of each HoneyFile. It is evident that attackers favor content that will give them immediate monetary compensation (such as PayPal accounts and credentials for our carding forum) than other data (e.g., email addresses and

[4]Carding is a term used for a process to verify the validity of stolen credit card data.

Filename	Claimed Content	Access Percentage
phished_paypal_details.html	Credentials to PayPal accounts	40.36%
card3rz_reg_details.html	Welcoming text to our fake carding forum and valid credentials	21.81%
Paypal_account_gen.exe	Application which generates PayPal accounts	17.45%
customer_list_2010.html	Leaked customer list from a known law firm	9.09%
Sniffed_email1.doc	Document with an embedded customer list of a known law firm	6.81%
SPAM_list.pdf	List of email addresses for spamming purposes	5.09%

Table 5: Set of files containing fake data that were used as bait in our HoneyFiles experiment. The third column shows the resulting download ratio of each file by attackers

Countries	Accesses
Russia	50.06%
Ukraine	24.09%
United States, United Kingdom, Netherlands, Kazakhstan, Germany	2.40% each
Sweden, Moldova, Latvia, India, France, Finland, Egypt, Canada, Belarus, Austria	1.20% each

Table 6: Attack geolocation recorded by our HoneyFile monitor

customer lists). Out of the 7 reported FHSs, one had a catalog functionality (listing all the uploaded files), two of them had a search option and the remaining four had neither catalog nor search functionality. Interestingly, one of the FHSs providing a search functionality did so through a different website, violating its stated Terms of Service (ToS). This shows that apart from abusing sequential identifiers attackers are also searching for sensitive keywords in FHSs that support searching.

Our monitor also recorded 93 successful logins from 43 different IP addresses at our fake carding website using the credentials that we included inside our HoneyFiles. When a valid username and password combination was entered, the carding website informed the user that the website was under maintenance and that she should have retried later. Fourteen out of the 43 attackers did so with the notable example of an attacker that returned to the website and logged in 14 times in a single day. The multiple successful logins show that attackers do not hesitate to make use of the data they find on FHSs. We assume that the fake PayPal credentials were also tried but we have no direct way to confirm our hypothesis.

In addition to login attempts, we also logged several attempts of SQL injection and file inclusion attacks conducted against the login page of our fake carding website and against our monitoring component. This shows that the attackers who downloaded the files from the vulnerable FHSs had at least some basic knowledge of web hacking techniques and were not plain users that somehow stumbled upon our HoneyFiles. We were also able to locate a post in an underground Russian forum that listed our fake carding website.

Monitoring Non-Sequential FHSs

For completeness, we decided to repeat the HoneyFiles experiment on 20 FHSs that adopt non-sequential identifiers. Interestingly, our monitor recorded 24 file accesses from 13 unique IP addresses originating from decoy documents placed in three separate FHSs over a period of 10 days. Upon examination, two of them were offering search functionality. While the third FHS stated specifically that all files are private and no search option is given, we discovered two websites that advertised as search engines for that FHS. Since our Honeyfiles could be found through these search engines and we never posted our links in any other website, the only logical conclusion is that the owners of that FHS partnered with other companies, directly violating their privacy statement.

We believe that the above experiments show, beyond doubt, that FHSs are actively exploited by attackers who abuse them to access files uploaded by other users. Unfortunately, since FHSs are server-side applications, the users have little-to-no control over the way their data is handled once it has been uploaded to the hosting provider.

5 Countermeasures

In previous sections, we showed that not only many file hosting services are insecure and exploitable, but also that they are in fact being exploited by attackers to gain access to files uploaded by other users. This introduces significant privacy risks since the content that users uploaded, and that was meant to be privately shared, is now in the hands of people who can use it for a variety of purposes, ranging from blackmailing and scamming to identity theft.

We notified 25 file hosting providers about the problems we found in our experiments. Some of them already released a patch to their system, for instance by replacing sequential IDs with random values. Others acknowledged the problem but, at the time of writing, they are still in the process of implementing a solution. Unfortunately, not all the vendors reacted in the same way. In one case, the provider refused to adopt random identifiers because it would negatively affect the performance of the database server, while another provider "solved" the problem by changing the Term of Service (ToS) to state that his system does not guarantee the privacy of the uploaded files.

Therefore, even though it is important to improve the

security on the server side, countermeasures must also be applied on the client side to protect the user's privacy even if her files end up in the hands of malicious users.

An effective way of securing information against eavesdroppers is through encryption, for example by using password-protected archives. In some cases, however, the default utilities present in operating systems cannot correctly handle encrypted archive files[5]. Therefore, we decided to design and implement a client-side security mechanism, *SecureFS*, which automatically encrypts/decrypts files upon upload/download and uses steganographic techniques to conceal the encrypted files and to present a fake one to the possible attackers. The motivation behind SecureFS is to transparently protect the user's files as well as providing a platform for detecting attackers and insecure file hosting services in the future (see Section 8).

SecureFS is implemented as a Firefox extension that constantly monitors file uploads and downloads to FHSs through the browser. When SecureFS detects that the user is about to upload a file, it creates an encrypted copy of the document and combines it with a ZIP file containing fake data. Due to the fact that ZIP archives place their metadata at the end of the file, a possible attacker who downloads the protected file will decompress it and access the fake data without realizing that he is being mislead. Even if the attacker notices that something is wrong (e.g., by noticing the size difference between the ZIP file and the resulting file) the user's file is still encrypted and thus protected. On the other hand, when a legitimate user downloads the file, SecureFS recognizes its internal structure and automatically extracts and decrypts the original file.

Most of the described process is automatic, transparent and performed without the user's assistance. The details of SecureFS and our prototype are publicly available[6].

6 Ethical Considerations

Testing the security of one hundred file hosting providers and extracting information for thousands of user files may raise ethical concerns. However, analogous to the real-world experiments conducted by Jakobsson et al. [5, 6], we believe that realistic experiments are the only way to reliably estimate success rates of attacks in the real world. Moreover, we believe that our experiments helped some file hosting providers to improve their security. In particular, note that:

- The enumerator tools accessed only partial information of the crawled files (the file's name and size) and did not download any file content.
- The enumerator tools employed a random delay between each requests to avoid possible impacts on the performance of the file hosting providers.
- We did not break into any systems and we immediately informed the security department of the vulnerable sites of the problems we discovered.
- The HoneyFiles were designed to not harm the user's computer in any way. Moreover, we did not distribute these files on public sites, but only uploaded them (as private documents) to the various FHSs.

7 Related Work

Different studies have recently been conducted on the security and privacy of online services. For example, Balduzzi et al. [2] and Gilbert et al. [16] analyze the impact of social-networks on the privacy of Internet users, while Narayanan et al. [10] showed that by combining public data with background knowledge, an attacker is capable of revealing the identify of subscribers to online movie rental services. Other studies (e.g., [13, 17]) focused on the security and privacy trends in mass-market ubiquitous devices and cloud-computing providers [11].

However, to the best of our knowledge, no prior studies have been conducted on the privacy of FHSs. In this paper we reported the insecurity of many hosting providers by experimentally proving that these services are actually exploited by attackers. There are, however, a number of cases where sequential identifiers of various services have been exploited. For example, researchers investigated how session IDs are constructed and in which cases they can be bruteforced [4]. The study is not restricted to IDs stored in cookies, but also analyzes the sequential and non-sequential IDs present inside URIs. Recently, researchers also identified issues with sequential identifiers in cash-back systems [15].

Antoniades et al. [1] notice that the recent increase of FHSs is threatening the dominance of peer-to-peer networks for file sharing. FHSs are found to have better performance, more content and that this content persists for a longer time than on peer-to-peer networks like BitTorrent. Researchers have also used FHSs as a distributed mechanism to store encrypted filecaches that can be used by a collaborating group of people [7].

Honeypots [12] have been traditionally used to study attacking techniques and post-exploitation trends. Yuil et al. [18] introduce Honeyfiles as an intrusion detection tool to identify attackers. Honeyfiles are bait files that are stored on, and monitored by, a server. These files are intended to be opened by attackers and when they do so, the server emits an alert. Similarly, Bowen et al. [3] use files with "stealthy beacons" to identify insider threats.

[5] How to open password-protected ZIP in Mac OS X, `http://www.techiecorner.com/833/`

[6] `http://www.securitee.org/sfs/`

8 Future Work

In Section 5 we presented our design for a client-side protection mechanism for files uploaded to FHSs. We described the process of appending a ZIP archive with fake data to each uploaded file.

While at the moment the fake document is a simple text file that informs the attacker of his wrong doings, we are currently investigating alternative uses. For example, the ZIP archive could contain a file that, when opened, "calls home" (see Section 4) and informs the file owner of the illegitimate file access. In order to preserve the user privacy, "home" can be a separate web service that will in turn report to users which file was maliciously downloaded and when. The service itself can also be adopted as a FHS monitor which will inform the users about vulnerable FHSs that should be avoided. This privacy-rating functionality can be beneficial to the community by allowing non-security inclined users to choose a secure FHS and by pushing the developers of the insecure FHSs to correct their privacy issues.

9 Conclusion

In this paper, we investigated the privacy of 100 file hosting services and discovered that a large percentage of them generate download URIs in a predictable fashion. Specifically, many FHSs are either vulnerable to sequential enumeration or they generate short identifiers that can be easily guessed by an attacker. Using different FHS enumerators that we implemented, we crawled information for more than 310,000 unique files. Using the Bing search engine as a privacy-classification mechanism, we showed that 54% of them were likely private documents since they were not indexed by the search engine. We also conducted a second experiment to demonstrate that attackers are aware of these vulnerabilities and they are already exploiting them to gain access to files uploaded by other users. Finally we presented *SecureFS*, a client-side protection mechanism which is able to protect a user's files when uploaded to insecure FHSs, even if the documents ends up in the possession of attackers.

Acknowledgements: This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, the IBBT, the Research Fund K.U.Leuven and from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 257007.

References

[1] ANTONIADES, D., MARKATOS, E., AND DOVROLIS, C. One-click hosting services: a file-sharing hideout. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (2009), ACM, pp. 223–234.

[2] BALDUZZI, M., PLATZER, C., HOLZ, T., KIRDA, E., BALZAROTTI, D., AND KRUEGEL, C. Abusing social networks for automated user profiling. In *Proceedings of the 13th international conference on Recent advances in intrusion detection* (Berlin, Heidelberg, 2010), RAID'10, Springer-Verlag, pp. 422–441.

[3] BOWEN, B., HERSHKOP, S., KEROMYTIS, A., AND STOLFO, S. Baiting inside attackers using decoy documents. *Security and Privacy in Communication Networks* (2009), 51–70.

[4] ENDLER, D. Brute-Force Exploitation of Web Application Session IDs. *Retrieved from http://www. cgisecurity. com* (2001).

[5] JAKOBSSON, M., FINN, P., AND JOHNSON, N. Why and How to Perform Fraud Experiments. *Security & Privacy, IEEE 6*, 2 (March-April 2008), 66–68.

[6] JAKOBSSON, M., AND RATKIEWICZ, J. Designing ethical phishing experiments: a study of (ROT13) rOnl query features. In *15th International Conference on World Wide Web (WWW)* (2006).

[7] JENSEN, C. Cryptocache: a secure sharable file cache for roaming users. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system* (2000), vol. 54, ACM, pp. 73–78.

[8] LAI, E. What's replacing P2P, BitTorrent as pirate hangouts? `http://www.computerworld.com/s/article/9139210/`.

[9] MUELLER, W. Top free file hosts to store your files online. `http://www.makeuseof.com/tag/top-free-file-hosts/`.

[10] NARAYANAN, A., AND SHMATIKOV, V. Robust de-anonymization of large sparse datasets. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 111–125.

[11] PEARSON, S. Taking account of privacy when designing cloud computing services. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing* (Washington, DC, USA, 2009), CLOUD '09, IEEE Computer Society, pp. 44–52.

[12] PROVOS, N. A virtual honeypot framework. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13* (Berkeley, CA, USA, 2004), SSYM'04, USENIX Association, pp. 1–1.

[13] SAPONAS, T. S., LESTER, J., HARTUNG, C., AGARWAL, S., AND KOHNO, T. Devices that tell on you: privacy trends in consumer ubiquitous computing. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium* (Berkeley, CA, USA, 2007), USENIX Association, pp. 5:1–5:16.

[14] SHARKY. 100 of the best free file hosting upload sites. `http://filesharefreak.com/2009/08/26/100-of-the-best-free-file-hosting-upload-sites/`.

[15] THIERRY ZOLLER. How NOT to implement a Payback/Cashback System. In *OWASP BeNeLux* (2010).

[16] WONDRACEK, G., HOLZ, T., KIRDA, E., AND KRUEGEL, C. A practical attack to de-anonymize social network users. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2010), SP '10, IEEE Computer Society, pp. 223–238.

[17] WRIGHT, C. V., BALLARD, L., COULL, S. E., MONROSE, F., AND MASSON, G. M. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 35–49.

[18] YUILL, J., ZAPPE, M., DENNING, D., AND FEER, F. Honeyfiles: deceptive files for intrusion detection. *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, June (2004), 116–122.