Deduplication as an attack vector

Marcus Einar Carl-Henrik Eriksson
Email: {marei265,carer706}@student.liu.se
Supervisor: Jan-Åke Larsson, {jan-ake.larsson@liu.se}
Project Report for Information Security Course
Linköpings Universitet, Sweden

Abstract

In this report we investigate if data deduplication in cloud storage services can be used as an attack vector. We investigate which of the cloud services are using data-deduplication and if they deduplicate across user accounts. We illustrate an attack on deduplication where a file is wrongfully deduplicated using the hash function MD5. We also discuss the likelihood of an attack where the much stronger SHA-256 function is used, and the limited brute force attack which relies on a limited set of changeable data (like a pin number) that can still be used to extract information even when the hash function is strong.

1. Introduction

Can data deduplication be used as an attack vector? That is a question we would like to find an answer to during this project. Data deduplication can be used in a broad spectrum of implementations. We have chosen to narrow it down to analysing how deduplication commonly works in cloud storage services, and whether it compromises file integrity and availability.

Several cloud storage providers use deduplication to save bandwidth and storage costs, and also to make the user experience better. In a common (yet simplified) implementation of deduplication, a hash-value of the file is calculated before it uploads. The hash is sent to the cloud and is used to check whether the file already exists. If it does exist, the file is not transmitted to the server again, and instead a reference to it is provided. Any of the providers may of course use other mechanisms in combination with hashes, to protect against exploits.

Our hopes is to find out which of the cloud storage services are using deduplication and if possible what hash functions are used to identify files. The later task has proven to be hard since most services are not open with their implementations.

To discuss this further we illustrate how attacks against file integrity and availability and confidentiality might be done by using hash collisions. We also try to shed some light on how probable a collision is using a hash function that is considered secure today. The results will be used to draw conclusions if it is feasible to craft attacks on file integrity and/or availability and the probability of collisions spontaneously occurring when a system has many files and users.

2. Background

This chapter covers a theoretical background on the concepts of data deduplication and cryptographic hash functions and how they are linked together. Attacks on hash functions and in turn deduplication are also introduced.

2.1 Data Deduplication

In general, data deduplication is a data compression method that makes sure that identical data is only stored once. This can of course save both bandwidth and storage costs, but may also come with privacy and security drawbacks. Data deduplication can be used across a variety of services, including mail providers, backup services and in cloud storage.

There are a couple of different implementations of deduplication. Firstly, we make the distinction between source-based and target-based deduplication. Source based deduplication happens at the *source*, i.e. in the application at the uploaders end. This saves bandwidth, but does nothing to hide the fact that deduplication has been used. In target-based deduplication it's the other way around, the file is uploaded before the deduplication occurs [2].

Deduplication can also be used on the file-level or block-level, meaning that an entire file can be deduplicated at once, or it can happen block by block (the size of the block varies in different implementations). The difference is that if two files exist on the server that are mostly the same, but not identical, deduplication can still be beneficial when it's block based [2].

It can also be implemented per-user, or cross-user. In the latter, redundant data is shared between accounts [2].

The common way to distinguish between different files or blocks is by using its hash value.

2.2 Hash Functions

In general, data deduplication is a data compression method that makes sure that identical data is only stored once. This can of course save both bandwidth and storage costs, but may also come with privacy and security drawbacks. Data deduplication can be used across a variety of services, including mail providers, backup services and in cloud storage.

There are a couple of different implementations of deduplication. Firstly, we make the distinction between source-based and target-based deduplication. Source based deduplication happens at the *source*, i.e. in the application at the uploaders end. This saves bandwidth, but does nothing to hide the fact that deduplication has been used. In target-based deduplication it's the other way around, the file is uploaded before the deduplication occurs [2].

Deduplication can also be used on the file-level or block-level, meaning that an entire file can be deduplicated at once, or it can happen block by block (the size of the block varies in different implementations). The difference is that if two files exist on the server that are mostly the same, but not identical, deduplication can still be beneficial when it's block based [2].

It can also be implemented per-user, or cross-user. In the latter, redundant data is shared between accounts [2].

The common way to distinguish between different files or blocks is by using its hash value.

2.2.1 Cryptographic Hash Functions

In addition to the properties of a hash function a cryptographic hash function must satisfy three more properties [1]:

The function must be **preimage resistant**, which means it should be hard to find a message with a given output. (Also called a one way function.)

The function must be **2nd preimage resistant** (or weakly collision free), which means that given any message it should be hard to find another message that gives the same hash.

The function must be **collision resistant** (or strongly collision free). This means that it should be hard to find any two inputs that result in the same hash value.

Here "hard" can be interpreted as infeasible with respect to the computational resources that is available at the time.

2.2.2 Common Cryptographic Hash functions

Since most file sharing services are not very open about their implementation we have had little success in discovering what hash functions are being used. Instead we decided to use MD5 and SHA-256 as a weak and a strong (respectively) cryptographic hash function for demonstrative purposes. We will use MD5 to illustrate how a broken hash function can be exploited and SHA-256 to attempt to quantify the probability and likelihood of an attack and/or hash collision.

2.2.3 MD5

Since most file sharing services are not very open about their implementation we have had little success in discovering what hash functions are being used. Instead we decided to use MD5 and SHA-256 as a weak and a strong (respectively) cryptographic hash function for demonstrative purposes. We will use MD5 to illustrate how a broken hash function can be exploited and SHA-256 to attempt to quantify the probability and likelihood of an attack and/or hash collision.

2.2.4 SHA-256

We use SHA-256 as an example of a secure cryptographic hash function. It is relatively secure as there are no known collisions that have been published to date of this report. It also has a message digest bit-length of 256 which is much more secure than the 128 bits of MD5. The SHA-2 family was published in 2002 [6] and approved as a standard in 2003 [7].

Even though weaknesses have been found in the algorithm [8] [9] we do not believe that it has any notable security impact at the current state since the weakness only applies to special cases where the rounds in the function are reduced.

2.3 Attacks on Cryptographic Hash Functions

There exists some attacks that are theoretically applicable to all hash functions. Three of them are discussed in the section below.

2.3.1 Brute Force Attack

A brute force attack is carried out by an attacker that tries to exhaust all possible combinations until a collision, preimage or second preimage is found. This is theoretically always possible but not applicable in practice when the message digest is long since in a worst case scenario the attacker has to try 2^n combinations, where n is the bit-length of the output.

2.3.2 Limited Brute Force Attack

Assume there are a limited set of possible preimages which are known to an attacker. The attacker also knows of a hash value and wants to know which preimage it belongs to, the attacker could calculate the hash value of all the preimages in the set until a collision is found. This attack is much more applicable in practice than the pure brute force attack since the worst case is equal to the number of preimages/hashes. This type of attack is useful against all types of hash functions where no other measures have been put in place, like digital signatures.

2.3.3 Birthday Attack

The name "birthday attack" are given to the attacks that is derived from the birthday paradox. The name describes a statistical (and maybe surprising) fact that if 23 people are in the same room the probability of two of them having the same birthday is over 50%. An approximation of the probability of r randomly selected values of N equally possible values, where N is large, could be calculated as [10]:

$$P_{match} \approx 1 - e^{-r^2/2N}$$

When looking for collisions in hash functions, r and N would correspond to r preimages and N possible digests. We will use the birthday attack later on to quantify the probability of hash collisions in deduplication.

2.4 Attack on deduplication using hash collisions

Where data deduplication is used, hash collisions could pose a great threat to file integrity, confidentiality and availability. We will use three examples to describe how this could work in theory. All examples will handle the case where two users are using the same file sharing service.

2.4.1 Example 1

User Alice (A) has a file F1 that hash the hash h(F1) = H and user Bob (B) has a file F2 that also has the hash h(F2) = H. Both users are using the same file sharing service that uses the function h(x) to get a fingerprint from files that are to be uploaded and stored. If deduplication is used, the following scenario might occur:

1. A tries to upload the file F1 to the server.
2. The server checks if h(F1) exists already. Since it doesn't, A uploads the file.
3. B tries to upload the file F2.
4. The server checks if h(F2) exists already. Since h(F1) = h(F2), the system thinks the file already exists, and B's file gets wrongfully deduplicated.
5. When B tries to download the file F2 from another client he will get the file F1 instead.

The described scenario shows how user B is unable to reach his file F2 which is a threat to the availability. In the case that the file F1 is a classified file this is also a threat to confidentiality. If the files were deduplicated at a block level and only a part of F1 and F2 had the same hash but contained different data, user B might end up with a corrupt version of F2 This would also deny B the availability of the file.

2.4.2 Example 2

In a similar scenario Alice (A) and Mallory (M) are using the same file sharing service. M knows that A is likely to upload a file F1 to the service at a future point. If M were able to forge a malicious file F2 so that h(F1) = h(F2) the following scenario might occur:

1. M uploads the file F2 to the server.
2. The server checks if h(F2) already exists. Since it doesn't, M uploads the file.
3. A tries to upload the file F1.
4. The server checks if h(F1) already exists. Since h(F1) = h(F2), A's file gets wrongfully deduplicated.
5. When A tries to access F1 at a later time she will get the malicious file F2 instead.

The described example shows how the integrity of a file can be compromised. This attack would be harder to carry out if files were deduplicated at a block level. Although the attack would still be valid if the size of F1 and F2 both were smaller than the block size.

2.4.3 Example 3

Eve applies the limited brute force attack as described in a previous chapter. Eve knows that Alice has uploaded a standard document containing a 4 digit PIN code to her file sharing service. Eve also knows that the document is created from a standard template where only the PIN is changed from document to document. Eve can now create different documents for all 10000 possible PIN code combinations (ranging from 0000 to 9999) and try to upload all of them to the file sharing server. If any one deduplicates, Eve will know that this is the document Alice received.

This shows how confidentiality could be compromised where the existence of a file and it's internal structure is known. This attack is valid in both the cases of file and block level deduplication.

This attack, however, is a bit naive since the document might contain signatures and/or other security barriers against such attacks, but naive attacks will work on naive implementations. However, the attack works independently on the implementation of the deduplication. The attack can view the system used in the attack as a black box.

3. Methods

3.1 Finding if deduplication is used

First we create a 10MB file with random data (to ensure the file is unique) using TrueCrypt, that we upload to the server. We then rename the file and upload it to a different folder. If the amount of data that uploads is significantly smaller than the file size, deduplication has occured. We then try the same thing again, but with two different accounts to see if the deduplication is implemented cross-user.

3.1.1 Which type of deduplication is it?

To find out which type of deduplication is used we can examine the tcp stream of an upload using, for instance, Wireshark. If we follow a single tcp stream we can immediately see if the transfer is done using blocks (and find a rough estimate of the block size) or if the entire file is transferred at once.

If it's block-based, we want to know the size of the blocks used. As stated previously, we can find an estimate of this size by examining the stream, but we also want to test this and see if it holds in practice. This can be done using the following method:

1. Create differently sized (unique) blocks of data (e.g. 2MB, 4MB, 6MB, 8MB)
2. Upload each file individually and measure the amount of data that actually transfers (should be the size of the file, plus some handshaking)
3. Concatenate each file with itself using the "cat" command, in order to create files twice as lage consisting of two identical blocks
4. Upload the new files and measure the data transferred

Relatively large files should be used (>1MB) to ensure that it can be seen in the traffic if it deduplicates or not. Since the traffic itself will be encrypted, smaller files will make it difficult to determine what has been transferred by only looking at the tcp packets.

3.2 Finding the services hash methods

This is hard to find evidence for, since all transmissions are encrypted. However, there was an open source project in 2011 called Dropship, which used SHA-256 to exploit deduplication in Dropbox by making it possible to download a file by only providing a hash [3]. Since then, Dropbox have implemented precautions against this kind of "attack", but it is unlikely that they changed their method of hashing, since that would do nothing to prevent Dropship-like applications.

There is no data on what hash-functions the other services use for deduplication.

3.3 Attack using hash collisions

3.3.1 Illustrated Attack using MD5 exploit

To illustrate how an attacker could craft two different programs with the same hash value, where one program is "good" and the other "evil", we will use a C-library called "evilize" developed in 2006 by Peter Selinger of Dalhousie University, Canada. This tool takes advantage of the block structure of MD5 and the technique developed by Xiaoyun Wang and Hongbo Yu [4]. The library contains two main programs:

- evilize: Can calculate an initialization vector and compile 2 programs with the same hash
- md5col: Finds a collision for the initialization vector

The process is done in four steps:

1. Write and compile the "good"/"evil" programs and link them to the "goodevil.o" file in the library
2. Use "evilize -i" on the compiled file to calculate the initialization vector.
3. Find a collision for the initialization vector using md5coll.
4. Build the program using evilize and the collision found in the previous step.

We will write two programs, both named "program", each of which will print a text message displaying if the program is "good" or "evil". This demonstration will illustrate how an attack could be carried out against a cloud storage service that deduplicates files using a hash function in which a flaw has been found.

3.3.2 Atack on SHA-256

SHA-256 has no known collisions to date, although it is theoretically weakened [8][9]. We will not try to use this weakness since it only applies to reduced versions of the algorithm. Instead we will try to apply previously mentioned attacks on cryptographic hash functions.

3.3.2.1 Brute force

We do not believe that SHA-256 is susceptible to brute force attacks since it is a relatively modern algorithm. We will try to quantify how hard it would be to use this method to find a SHA-256 collision.

3.3.2.2 Bithday Attack

There might exist a birthday attack if a cloud storage provider has many users which in their turn store many unique files. We will use the birthday attack method described in section 2.2.3 to calculate how many preimages it would take to have a 50% chance of a collision.

Dropbox offers their users to automatically backup their photos from their desktop or mobile applications. Such sync features could possibly generate a lot of unique files. Given that dropbox now have over 275 million users [11] and all of them probably have some unique files each there might very well exist a birthday attack. An interesting number in such a case would be how many files per user gives a 50% chance of a hash collision.

4. Results

In this section we present the results of how deduplication is done and what hash functions are used in the cloud storage services and what weaknesses they present.

4.1 Deduplication

Using the method described in section 3.1 we have examined the following providers:

- Dropbox
- Google Drive
- OneDrive
- Amazon Cloud Drive
- SpiderOak

- Box
- Memopal
- AltDrive
- Wuala

4.1.1 Dropbox

In diagram 1 below we can clearly see that the 4MB and 8MB blocks deduplicates "perfectly", while the 6MB block benefits from some deduplication, which means that Dropbox use block-based deduplication with a block size of 4MB.

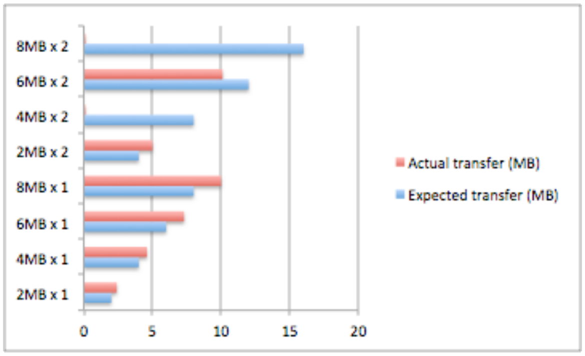

Diagram 1. Expected transfer size vs. Actual transfer size for Dropbox and Wuala

Note that there is a margin of error in the size of the transmission due to hand shaking and occasional retransmissions. If no deduplication is used, the actual transfer size should be slightly larger than the expectation, which it is.

While the files are still stored on one account, we try to upload some of the files to a different account to see if deduplication occurs cross-user. In diagram 2 we can clearly see that this is not the case. If dropbox uses cross-user deduplication, they hide this from the users.

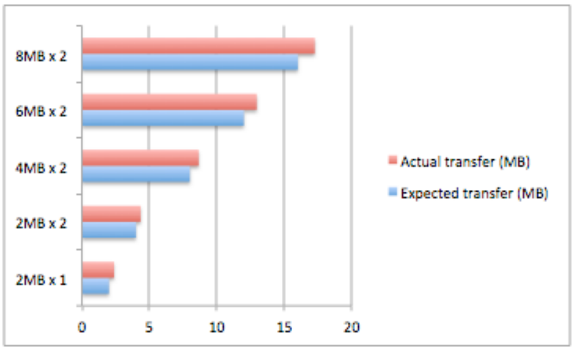

Diagram 2: Expected vs actual transfer size between accounts

4.1.2 SpiderOak

When the same file (but with a different name) is uploaded more than once, the total uploaded data doesn't exceed 20KB for any given file. When we examine different block sizes we see no deduplication at all (diagram 3). This is also supported by the fact that when we examine the tcp stream of a file transfer, we see the whole file upload at once. This means SpiderOak deduplicates data at the file-level. They don't, however, deduplicate data cross-user (see diagram 2).

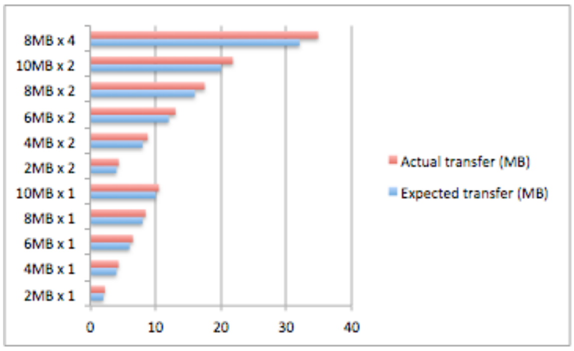

Diagram 3: Test for block-level deduplication in SpiderOak/Memopal/AltDrive/Wuala

4.1.3 Memopal

Memopal is one of the only two services we've found that deduplicates files cross-user. When a file is uploaded first to one account, and then to a second account, the second transfer is nearly instantaneous. They implement deduplication at the file-level (See diagram 3).

4.1.4 AltDrive

When testing for deduplication of individual blocks, we see no deduplication at all (see diagram 3). AltDrive deduplicates at the file-level. They do not deduplicate cross-user (see diagram 2).

4.1.5 Wuala

Wuala is, alongside Memopal, the other service that deduplicates files cross-user. Unlike Memopal, they use block-level deduplication with a block size of 4MB (see diagram 1).

4.1.6 The other services

Google Drive, **OneDrive**, **Amazon Cloud Drive**, and **Box** isn't using any type of source-based deduplication. If the same 10MB file is uploaded twice (to different folders, and with a different name), 10MB of data will be transferred both times. This means that if they are using deduplication at all, they are hiding it from the user, and we can do nothing to examine it.

4.2 Hash function weaknesses

Here we discuss the weaknesses of different hash functions and what possible weaknesses they present. We also mention something about the probability of collisions in the hashes.

4.2.1 Illustrative attack using MD5 exploit

The illustrated attack as described in chapter 3.3.1 was successful. The most time consuming step was to calculate a collision for the initialization vector, which took about an hour on a Macbook from 2009 but can sometimes take up to several hours on the same hardware. Both program files created share the same MD5 hash-value *"b402794a9890eeb6898cc519110bb58b"* and name *"program"*. When the good program is executed it prints a text stating that it is a good program:

*"**$ ./good-files/program***
This is a good program created for TDDD17 by:
Carl-Henrik Eriksson
and
Marcus Einar
(press enter to quit)".

The evil program states that it indeed is a program meant to harm the user who executes it:

*"**$ ./evil-files/program***
This is an 3V1L program created for TDDD17 by:
Carl-Henrik "H4kkZ0r" Eriksson
and
Marcus "M1sChi3V0us" Einar
(press enter to quit)"

This example of an attack shows that where a weak hash function is used it is very possible for a malicious party to create harmful files that share the same hash values with another not-harmful file.

4.2.2 Attack on SHA-256

4.2.2.1 Brute force and limited brute force

Since SHA-256 has an output of 256 bytes, that is 2^{256} possible values, brute forcing is simply not possible. As a comparison there are approximately 10^{50} atoms on the earth, according to Wolfram Alpha, and one of the fastest hashing computers we have found can calculate approximately 348 billion hashes per second[12]. If every atom on earth was such a computer it would still take over 100 million years to calculate all possible hashes.

A limited brute force attack would depend on the number of possible preimages and not the possible output values of SHA-256 and if other security barriers have been put in place.

4.2.2.2 Birthday attack

When calculating a birthday attack probability on SHA-256 with 2^{256} possible different combinations of outputs it would take approximately 4.00625×10^{38} preimages to get a probability of 50% of a spontaneous collision. This would mean that every user in a system that, like Dropbox, has 275 million users would have to own more than 1.45×10^{30} files each. This corresponds to more files than the estimated content in the web as of 2001 (according to Wolfram Alpha) per user if all files were one bit small.

Since these numbers are too high to ever be realised we have tried to approximate how likely a collision actually is. If we assume that every user has at least 5000 unique files and there are 275 million users like in the dropbox case we get the approximate probability of 10^{-70}. This shows that a spontaneous collision is highly unlikely to occur.

5. Conclusions

Our investigation of different cloud storage services show that the majority of them does not use cross-user deduplication. This might be due to the controversy about Dropbox privacy concerns when they were using it in their service, and we were surprised when we found that they no longer implement a cross-user strategy. We think they might have dropped this approach after the successful exploit "Dropship" [3] was published which might have given them concerns of facing future lawsuits from violation of copyrighted files.

Our calculations on the currently strong hash function SHA-256 show that it is very unlikely that collisions will occur spontaneously where cross-user deduplication is used. It is even more unlikely that a malicious party would be able to use deduplication as a vector to infuse malware into someone elses file systems. We believe that the integrity and availability of the files are relatively safe at the current state.

Special cases of SHA-256 is has been weakened, but it is far from broken in the full implementation. However, what is considered safe today might be proven unsafe tomorrow, and "attacks never get worse; they always gets better" is a somewhat famous quote when speaking about security. There is also the risk of unpublished attacks that work from either malicious hackers, organisations or governments. There is also a risk of there being backdoors that are built in. It is important to always stay in the loop when it comes to security, and when SHA-256 gets closer to being broken, cloud services need to be ready to update to a safer method of hashing.

The limited brute force attack can still be used regardless of hash-function. In naive implementations naive attacks will work which means that further security measures needs to be taken to guarantee security.

6. References

[1] A. Menezes, P. van Oorschot and S. Vanstone, "Handbook of Applied Cryptography"

[2] D. Harnik, B pinkas, A Shulman-Peleg, “Side Channels in Cloud Services: Deduplication in Cloud Storage”, 2010
[3] W. van der Laan, “Dropship”, 2011 (https://github.com/driverdan/dropship)
[4] X. Wang, H. Yu, “How to break MD5 and other hash functions”, 2005
[5] R. Rivest, “RFC 1321”, 1992 (http://tools.ietf.org/html/rfc1321)
[6] “FIPS180-2”, 2002 (http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf)
[7] “Approval of FIPS180-2”, 2003 (https://www.federalregister.gov/articles/2002/08/26/02-21599/announcing-approval-of-federal-information-processing-standard-fips-180-2-secure-hash-standard-a)
[8] D. Khovratovich, C. Rechberger, A. Savelieva, Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family, 2011 (http://eprint.iacr.org/2011/286.pdf)
[9] M. Lamberger, F. Mendel, “Higher-Order Differential Attack on Reduced SHA-256”, 2011(http://eprint.iacr.org/2011/037.pdf)
[10] W. Trappe, L.C. Washington, “Introduction to Cryptography with Coding Theory”, 2006
[11] A. Wilhelm, “Dropbox Hits 275M Users And Launches New Business Product To All”, 2014 (http://techcrunch.com/2014/04/09/dropbox-hits-275m-users-and-launches-business-product-to-all/)
[12] P. Roberts, “New 25 GPU Monster Devours Passwords In Seconds”,2012, (https://securityledger.com/2012/12/new-25-gpu-monster-devours-passwords-in-seconds/)