:: Snerx Exploit Reification ::

Matthew R. Garon - 2018/4/26

Sploitation Documents

INDEX:

:: Hash Reversal with IPFS ::

The internet is still very much the Wild West, don't be fooled. Exponential growth and new technologies that change the underlying protocols used for data transmission means new and exponential increases in the number of flaws to exploit. This section outlines a system you can easily game, with no first-mover requirement. It would take years to steal all the money and assets in this proposed system so it doesn't matter if many other people are exploiting it as well. No depth is unreachable.

To be clear, we've already tested this, it works, it's arbitrary to execute, and it gives you a god-throne to sit on. In this ocean of potential, we can be pirate kings. Any cries of 'legality' can be dismissed as *pro bono publico* pen-testing, OpSec-awareness, defense analysis, or any other of the myriad escape routes that legitimize use. Put on your boots and stomp the sun-kissed planks of this glorious pirate ship with us. We can raze the ports, set the seas ablaze, and be all but burnt ourselves.

General points on IPFS - The Interplanetary File System uses self-signing multihash; this means most major hash algorithms and their hex, base32, base58, and base64 variants are recognized.[1] You can generate a SHA hash of a file in your terminal and pipe it to IPFS, telling it to crawl for that hash; this means you can return hash content instantly as long as it exists on IPFS already.[2] IPFS is an infinitely-scalable hash-archive and it's growing exponentially right now, matching the exponential rate of data expansion and storage we've been seeing since the early 90's.[3]

Steganography with IPFS - You can use IPFS to hide messages so long as the relevant hash was generated and pinned somewhere on IPFS. To implement, take your plaintext message and save it as a file in your IPFS directory, then send the corresponding hash as the direct message. You can also use this to hide images or video; any type of file can be hosted on IPFS, finding and accessing the file is arbitrary if you know the hash. There are obvious additional implementations of steganographic content this way, but as far as I know, there is currently no standard thing called 'hash-based steganography' since hashes are not supposed to be reversible. Unless I am unaware of something obvious, this is a first.

Password cracking with IPFS - Generating hash lookup tables from string-permutations in order to break into databases whose users' passwords were hashed would normally take a lot of time. Even with rainbow tables the lists are many terabytes or petabytes of information. However, since IPFS is

[1] Multihash's git repository and their full list of supported hashcodes.
[2] And for future reference, you can see if any peers are hosting a file with: $ ipfs dht findprovs [hash]
[3] A formal paper on this and an article written in 1997.

essentially a distributed hash lookup table,[4] you could generate a complete lookup table *once* and make sure it gets distributed/pinned, creating a permanent uncensorable hash lookup table. Petabytes of possible passwords could be stored off-site, and currencies like FileCoin will ensure people are willing to host these IPFS-based archives.[5] The disproportionate majority of hashes from passwords used by humans could be instantly looked up and their corresponding plaintexts returned with them. To keep this feasible, you would cap the hash generation to password permutations *N* characters long (whatever arbitrary amount you decide on). This does not get *all* passwords, strong passwords would stay secure, but it will get you *most* passwords used for almost every site. Something something *ipso facto* now you have access to billions of dollars worth of assets, currencies, conversations, blackmail, and more.[6]

A full Latin alphabet, base-10 numbers, and 55 additional standard characters are contained within a one-byte space in UTF-8 character encoding.[7] Total, there are 107 relevant characters that show up on an English keyboard that we would use as our base charset for bruteforcing. The formula being used is:

(C^L)*B = bytes by decimal

Where C is the size of the character set being used, L is the length of the string being permutated, and B is *bytes*, which in the case of this bruteforce attack is the same number as the length of the string.

(107^9)*9 = 16,546,132,911,781,390,563	16 exabytes
(107^8)*8 = 137,454,894,386,553,608	137 petabytes
(107^7)*7 = 1,124,047,033,534,901	1.1 petabytes
(107^6)*6 = 9,004,382,111,094	9 terabytes
(107^5)*5 = 70,127,586,535	70 gigabytes
(107^4)*4 = 524,318,404	524 megabytes
(107^3)*3 = 3,675,129	3 megabytes
(107^2)*2 = 22,898	22 kilobytes
(107^1)*1 = 107	107 bytes
Total from 1-7 = 1,133,122,071,249,068	1.1 petabytes

Permutations 8-long and 9-long generate too much data for us, so we will be using 7 and lower. Using a reduced character set of only upper and lowercase alphanumeric characters (62 total) we can generate slightly longer permutations.

(62^10)*10 = 8,392,993,658,683,402,240	8 exabytes
(62^9)*9 = 121,833,778,916,371,968	121 petabytes
(62^8)*8 = 1,746,720,844,679,168	1.7 petabytes

So for generating a bruteforce table we'll only be reliably catching passwords up to 7 characters long, and additional weak passwords up to 8 characters long. Most normie passwords are about this length, which means we've already caught the majority of passwords. However, many websites now require passwords to be at least 8 characters and use special characters to resist bruteforce attacks. This is a poor security measure since humans don't remember passwords like *Tr0ub4dor&3* very well, but we do

[4] IPFS's main documentation.
[5] FileCoin is a cryptocurrency started by Protocol Labs, the same developers of IPFS.
[6] This was originally written about here.
[7] https://www.fileformat.info/info/charset/UTF-8/list.htm

remember *correct horse battery staple* very easily even though it has significantly more entropy.[8] Better security practices require longer word-based passwords like the one just referenced rather than shorter random-character passwords. This is due to the significantly higher entropy and therefore significantly higher processing power needed to bruteforce these passwords. In the referenced article, it is noted that an 11-character-long password still only takes around 3 days to bruteforce despite the extended character set. We can now cut that time down a little bit due to all hashes being archived up to at least 7 characters on IPFS. But this is still not good enough, we would want to be able to catch passwords significantly longer for obvious reasons, and so we will also be generating and uploading a lookup table using a dictionary attack as well. The formula being used is:

$$(D^L)*B = \text{bytes by decimal}$$

Where D is the number of words and assorted characters in the dictionary, L is the number of terms being permutated at a time, and B is *bytes*, which in this case we're rounding to around 30. The reason for bytes counting as 30 is because *correct horse battery staple* is only 28 characters long and the average word length in password dictionaries is 6 characters long.

(5000^5)*35= 109,375,000,000,000,000,000	109 exabytes
(4000^5)*35= 35,840,000,000,000,000,000	35 exabytes
(4000^4)*30= 7,680,000,000,000,000	7.7 petabytes
(4000^3)*25= 1,600,000,000,000	1 terabyte
(4000^2)*20= 320,000,000	320 megabytes
(4000^1)*10= 40,000	40 kilobytes
Total from 1-4 = 7,681,600,320,040,000	7.7 petabytes

Using a dictionary with 5,000 terms was way too big, so we dropped it to 4,000 and got a significantly smaller storage output. We lowered it from five terms to four terms as well since most word-based passwords are only three or four terms long. This leaves us with a feasible storage output. The main public Diceware list includes 7,776 terms (5 6-sided dice is 6^5), and about half are non-word character combinations, so a 4,000-term dictionary is more than enough for most word-based passwords.[9]

Before we talk about costs, it should be pointed out that these are just starter tables. They can be expanded and added on to arbitrarily through IPFS. As Caringo's Swarm overview notes, "90% of the data in the world has been generated in the last 2 years."[10] And as was said earlier in this document, file storage is exponentially growing; the costs of this storage is cut in half every year. This means that while currently 35 exabytes is way too expensive, in two years time it will be feasible for a large organization to buy this amount of storage space.

All tables in this document totaled equal around 10.5 petabytes, which is feasible. Through generic cloud storage options like Storj, you can get around a gigabyte of storage for 4 cents USD.[11] If we were using cloud storage, this would equate to around $420,000 USD for 10.5 petabytes of information. Given that this halves every two years, we can estimate a gigabyte to cost 1 cent USD in two years time, making the cloud storage option cost $105,000 USD for 10.5 petabytes in two years.

FileCoin uses IPFS, which scales far more frictionlessly than cloud storage, will be far more competitive than proprietary corporate options, and also allows for ask-pricing of users far lower than 1

[8] Forgive me for referencing this dumb web comic.
[9] https://world.std.com/~reinhold/diceware.wordlist.asc
[10] https://www.caringo.com/swarm-overview/
[11] https://storj.io/

cent USD,[12] so the estimated cost of 10.5 petabytes on IPFS is around $1,050 USD given a gigabyte pricing of 0.0001 USD. Even if this price turns out to be currently unrealistic, it won't be unrealistic in two years' time. This means you can have entry into this market, this ocean of potential, for about a thousand bucks. Further, only one person needs to drop this rack since IPFS is totally open and permanent; all files on it can be accessed by anyone and is incredibly difficult to censor.

There are alternative networks to IPFS, but appear to be inferior both in storage capacity and functionality. The two main alternatives are Swarm and Sia. A good comparison between IPFS and Swarm can be found here, where IPFS is the clear winner. Sia's total storage capacity, as shown on their landing page today, is only 4.8 petabytes. That is tiny compared to IPFS, whose FileCoin poll showed many people looking to rent over 10 petabytes at a time.

In a comical world, all alternatives would be used concomitant to IPFS, artificially inflating the price of all the cryptocurrencies backing each, but I doubt the world will come through for us on this one. So instead, we will only be artificially inflating the price of FileCoin; buy it while you can.

Cracking private keys in asymmetric-key protocols with IPFS - In short, this cannot be done. If IPFS can be used as a giant hash table archive, then you could just as easily use it as a private-key archive, right? E.g. we could look at Bitcoin addresses. You can generate all public and private key pairs already through services like directory.io and run tests with gobittest, but generating the full list as a single instantiate is infeasible, regardless the network hosting it. Even if IPFS becomes as large as the web, there isn't enough storage space on all the disks of the world to hold even a substantial minority percentage of the address pairs.

If there was enough space on IPFS to hold 0.1% of the key-pairs at a time, then you could run an exploit that chugs through that fraction of a percent at a time, only saving keys whose public addresses have coinage sitting on them, right? Processing all this is infeasible still. Short lists are arbitrary to generate, but long lists would be limited by the speed at which you could check the addresses and have the relevant ones uploaded and mirrored through IPFS, which would be incredibly slow. Even without having to upload to IPFS, it would still take thousands of years to generate the key pairs in the first place, let alone perform any actions with them all.

Unless there is a way to make the linked directory reverse-searchable so it isn't ordered just by private keys but also by public keys, then there is no feasible way to find key-pairs. All that being said, this was just as much of an outline on the potential of IPFS as it was the insecurities of hashcodes. My hopes with carrying this exploit out is to expose that hash-based security isn't as secure as people generally believe, and that for the two most common password-types (showcased earlier), the disproportionate majority of those passwords can now be very easily broken (instantly in fact). Hopefully this helps people move towards other methods of securing data that don't rely on hashcodes.

If someone has beaten me to all this already, or if I'm missing something obvious and devastating, I would like to be corrected, thanks - snax@snerx.com

[12] https://filecoin.io/filecoin.pdf