:: Diogenesis' Exploit Reification ::

Created 2018/4/26 | Updated 2019/11/26

Sploitation Documents

INDEX:

• Postmates Cartel: when *free as in free beer* means free beer literally and for all other purchases.
• Hash Reversal: using distributed lookup tables to eliminate the need for bruteforce attacks.
• C&C Forums: automated sublimation and implosion of online communities using many faces.
• Tempest Monitoring: the great indoors; wardriving for keystroke and visual capture.

:: Introduction ::

The internet is still the Wild West. 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. We have tested and implemented all the exploits we cover here; they work, they're arbitrary to execute, and they give you a lot of room to experiment with. In this ocean of potential, we are pirate kings. Any cries of questionable 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.

:: Postmates Cartel ::

Postmates is a courier service that delivers any product listed in their app to your doorstep. Postmates also has broken company internals. They are terribly ran, which allows room to exploit them. If you ever wanted free shit, this is how you get free shit. Postmates has written about one of us doing this.[1] A quick outline of how the execution occurs is as follows:

> You need an iPhone with Apple Pay, the Akimbo Card app, and the Postmates App (3.8.0 or older). You need to know which department you're going to hit; the location and item number. Lastly, you need a Postmates account under a fake name and tossable email address. Attach the Akimbo Card to your Apple Pay card then order an item and wait outside the department store you ordered it from. Pick it up from the Postmates courier in their parking lot, and tip them $99 in the app to build trust. Finally, delete your account, reset the iPhone or toss the burner and get a new one with the money you made. If it gets canceled simply reset the phone and repeat the prior steps.

Why the older version of Postmates? Standard delivery apps such as Uber Eats or Grub Hub charge the full amount on debit/credit cards *before* the delivery is made. But Postmates was made by a team of retards, so they charge your account *after* the delivery is made and the product is in your hands. As a 'safety' measure, Postmates has a preauthorization charge of one cent which does not go through. This fake charge only occurs in the older versions of Postmates (all versions up to 3.8.0). The newer versions all have a one dollar pre-authorization charge that *does* go through. Getting the older version

[1] https://blog.postmates.com/fighting-fraud-with-data-de92680a384

will allow your Akimbo Card to go undetected so you won't need any money on the card to execute the exploit. You could also just put a couple dollars on the card to avoid this hassle.

Why Akimbo? After trying several card apps for this exploit, Akimbo ended up being the easiest and most effective. You can create multiple sub-cards on your primary card as well as create new cards in under five minutes. You can also easily attach them to Apple Pay, which is necessary because this is the only way Postmates will accept your payment. Apple Pay encrypts the fake info you put in your Akimbo Card which adds an extra layer of security for the user running this exploit. Other card apps such as Square Cash work as well but they aren't as effective. The ease of which Akimbo lets users do all of this makes it seem like Akimbo was designed for fraud.

How do you start ordering? After creating a Postmates account with false credentials, set up your Akimbo Card with the same false credentials and attach it to Apple Pay for obfuscation. The next step is to find what item you want and which department store you would like to order it from. Best Buy, Wal-Mart, and Target are the easiest department stores to get electronics and high-price items.

A note with this exploit - if your order exceeds two-hundred dollars, it will be live-tracked and your chances of getting what you ordered decrease dramatically. Maximizing your success depends on how close you are to the department store from which you ordered and being as discreet as possible in revealing what the price of your item is. For example, instead of ordering by typing 'MacBook', use the SKU number (like 4260600). The Postmates backend doesn't make time to track your item unless they directly know how much the item costs, and by using the SKU number Postmates doesn't know the cost of the item until the courier is swiping their card. Items under two-hundred dollars do not get tracked at all, which leaves food, booze, and low-end items not only free but available for delivery no matter how far you are from the store of their origin.

After inputting the SKU number, you are ready to order your item. An effective strategy to use alongside SKU numbers is to sit outside the department store's parking lot and catch your courier as they exit the building. When the courier is confronted, they will suspect something is off since you are waiting in the parking lot of the place you just ordered delivery from. Assure them that the only reason you ordered from Postmates was to use the free hundred-dollar credit they give you. Don't spend too much time with them, no need to let them memorize your appearance, just handoff the items.

Put the items in your trunk so they aren't visible if you're pulled over and tip the courier the maximum possible of ninety-nine dollars. This builds rapport with the couriers in your area and makes them less likely to talk. Make sure to reset your phone and use the money you just made to buy a new burner with Apple Pay so you can sell the burner on Offer Up after each successful delivery on high end items for extra profit. Bonus points for backdooring the burner after reset and surveilling the people that purchase pre-used burner phones in your area.

Through extended use of the process, lots of rinsing and repeating, you can build a cartel around this exploit by getting Postmates' couriers to work for you directly and deliver free shit in large quantities. The following is a story from the Diogenesis Table Society that took place over the course of three months which achieved this exact feat.

My first delivery was made in Las Vegas when I ordered fifty burgers for my friends from White Castle. I thought the app would reject the service request but it didn't. We had free food for the entire weekend. A month after discovering unlimited access to food and drinks, I decided to go for something bigger. I went to a Best Buy in San Diego and parked my car outside their lot, entered the order for a MacBook with the SKU of 4260600, and it arrived fifteen minutes later. Fifteen minutes, twenty-four hundred dollars. It was time to buy burners and see how many MacBooks I could go through.

A couple days later I ordered two MacBooks at the same time and had Postmates deliver them to me back to back. Forty-eight hundred dollars. I had setup with a buyer on Offer Up who would pay

me cash for the MacBooks, no questions asked. I would give them to the buyer slightly under their retail price and he would send them to his international reseller at profit. Thus I began regularly making trips to different Best Buy's and ordering MacBooks about twice a week. I would tip the drivers the maximum allowed by the app after my orders were completed, ninety-nine dollars, because I wanted to build rapport with them in case questions were asked about me.

Loyalty incentivisation through tipping worked. One day after a free MacBook delivery, one of the Postmates' couriers that I knew warned me about a message sent from their corporate security team regarding a fraudster in the area. I was the fraudster. The courier knew they were talking about me, but my ninety-nine dollar tips had earned me a friend and persuaded him not to report me.

As a cooldown, I stopped orders for a month. Postmates released an article about how they were improving their security and how they were tracking fraudsters during this cooldown[2], which was nice of them because it let me know I would now have a limited amount of time to execute an order before their tracking team could catch and cancel it. I pulled a fake order to test how fast their fraud prevention team would be able to track and cancel it with their new system. They caught it ten minutes in, which was just about how long it took a courier to get to the store, meaning the exploit was no longer doable. I needed to drastically cut down execution time and I had an idea - why not employ my own couriers?

I reached out to some friends, asked them if they wanted to make fast money, and started them on legitimate orders to cover up the fraudulent ones. They would run about a month of legitimate deliveries for normal users of Postmates, then I would take them to a Best Buy, have them wait inside with a MacBook at the counter, and I would ping them in the app for an order.

This got a little messy since the Postmates courier card has a limit of two-hundred dollars. The courier would have to add funds immediately after each swipe and swipe again many times over. As mentioned earlier, the MacBooks I would order cost around twenty-four hundred dollars including tax, so I told my couriers to be ready to swipe exactly twelve times. I began to chart how much time it would take for them to swipe and get to the correct amount, the average being around 3 minutes for the swiping process and 7 minutes total for them to go into the store and come back out with the product. We consistently beat the Postmates tracking team with plenty of time to spare.

This grew pretty fast. My friends spread the word about the easy money, and people came to me asking to work for me. At its height, this was a system of just under twenty couriers who were ready at any time to go to a store and walk out with armfuls of MacBooks. Tens of thousands of dollars in cash on top of free meals and vacations were flowing through my Postmates cartel and everyone got a piece.

Then I quit. I got what I wanted, and I didn't want to get arrested, so I quit. Everything was burned, and everyone walked away unscathed. The statute of limitations puts me safely outside of legal repercussion, but an affinity for this kind of work rarely dies easy.

[2] Ibid.

:: Hash Reversal with IPFS ::

This section outlines a system you can enter with ease, 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 already exploiting it as well. No depth is unreachable.

General points on IPFS - The Interplanetary File System 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] IPFS uses self-signing multihash; this means most major hash algorithms and their hex, base32, base58, and base64 variants are recognized.[4] 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 already exists on IPFS.[5]

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 the first robust implementation of hash-based steganography.

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 essentially a distributed hash lookup table,[6] 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.[7] 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* or even *most* passwords, strong passwords would stay secure, but it will get you *many* passwords used for any site. Something something *ipso facto* now you have access to billions of dollars worth of assets, currencies, conversations, blackmail, and more.[8]

A full Latin alphabet, base-10 numbers, and 55 additional standard characters are contained within a one-byte space in UTF-8 character encoding.[9] 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:

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

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.

---

[3] A formal paper on this and an article written in 1997.
[4] Multihash's git repository and their full list of supported hashcodes.
[5] And for future reference, you can see if any peers are hosting a file with: $ ipfs dht findprovs [hash]
[6] IPFS's main documentation.
[7] FileCoin is a cryptocurrency started by Protocol Labs, the same developers of IPFS.
[8] This was originally written about here.
[9] https://www.fileformat.info/info/charset/UTF-8/list.htm

(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. Many passwords are weak and about this length, which means we catch a lot of passwords here, but many websites require passwords to be at least 8 characters and use special characters to resist bruteforce attacks. Humans don't remember passwords like *Tr0ub4dor&3* very well, but we do remember *correct horse battery staple* very easily even though it has significantly more entropy.[10] Better security practices require these 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, 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 = 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 approximately 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

[10] Forgive me for referencing this dumb web comic.

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 standard 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.[11]

An aside; many major cryptocurrency wallets have deterministic word-seeds used to generate their key-pairs and all of these wallets are open source, meaning the dictionaries they use are known and the term lists are public. You can use the prior equation to determine how large a complete table of permutations would be for these wallets. Many dictionaries are only 1,000 words large, but the standard word-seed length is 12 terms, which gets us something clearly too large to store:

(1000^12)*70 = 7×10^{37}	60,715,321,659,188,248,304.42 exabytes

As we start to 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."[12] And as was said earlier in this document, file storage is exponentially growing; the costs of this storage are cut in half every year. This means that while currently 35 exabytes is way too expensive, in two years time it will be arbitrary for an 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.[13] If we were using proprietary cloud storage like Storj, 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 four years time, making the cloud storage option cost $105,000 USD for 10.5 petabytes.

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 cent USD,[14] 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 early.

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

[11] https://world.std.com/~reinhold/diceware.wordlist.asc
[12] https://www.caringo.com/swarm-overview/
[13] https://storj.io/
[14] https://filecoin.io/filecoin.pdf

archive, right? E.g. we could look at Bitcoin addresses. You can generate all public and private key pairs already through services like keys.lol and run tests with gobittest, but generating the full list as a single instantiate is infeasible, regardless what network is 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 this would still be infeasible. 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.[15] 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 currently no feasible way to find key-pairs.

All this considered, there are already hash databases around, with hashes.org being the most notable one. So why bother with this exploit? Well, the hashes.org database can be mirrored onto IPFS, removing the burden from a local server, and it can then be expanded on arbitrarily at exponentially lower costs than what their current hosting rate is. We are looking to use IPFS as a more open version of hashes.org. This was just as much of an outline on the potential of IPFS as it was the insecurities of hashcodes. For the two most common password-types from 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, or at least improve hash systems such that they can't be cracked in static sets of time.

[15] Schneier, Bruce. *Applied Cryptography: Protocols, Algorithms and Source Code in C*.