Cyclone https://github.com/cyclone-github
Flagg https://github.com/flaggx1
URL https://forum.hashkiller.io
Date Dec 30, 2024, rev 2

Thanks to Hashkiller, Hashes, and Freeroute for this generous $1,000 Holiday challenge. Although parts of the challenge were extremely difficult, hints were provided in the forum post as the event went on, leading to a successful outcome.

Part 1

The challenge started with a file attachment:

hashes_com_holiday_hashmas_puzzle_2024.zip.

This archive expanded into the following files:

```
riddle.txt
merry_christmas.png
hint.txt
part_2/part_2.zip
```

The **riddle.txt** file contained a rhyme with clues:

```
Four keepers guard the vault you desire,
Their essence hidden in a digital fire.
Each with a secret, both front and back,
Choose wisely now, there's no turning back.
Combine their strength, first or last,
To form the key that will hold you fast.
A 32-character code is what you need,
Crack this riddle, and you'll succeed.
```

The **hint.txt** contained a 32-character hex string which we assumed was MD5 or another similar algorithm: At some point on the forum the letters “M”, “D”, “5” were bolded in a hint which solidified that we weren’t working with another algorithm.

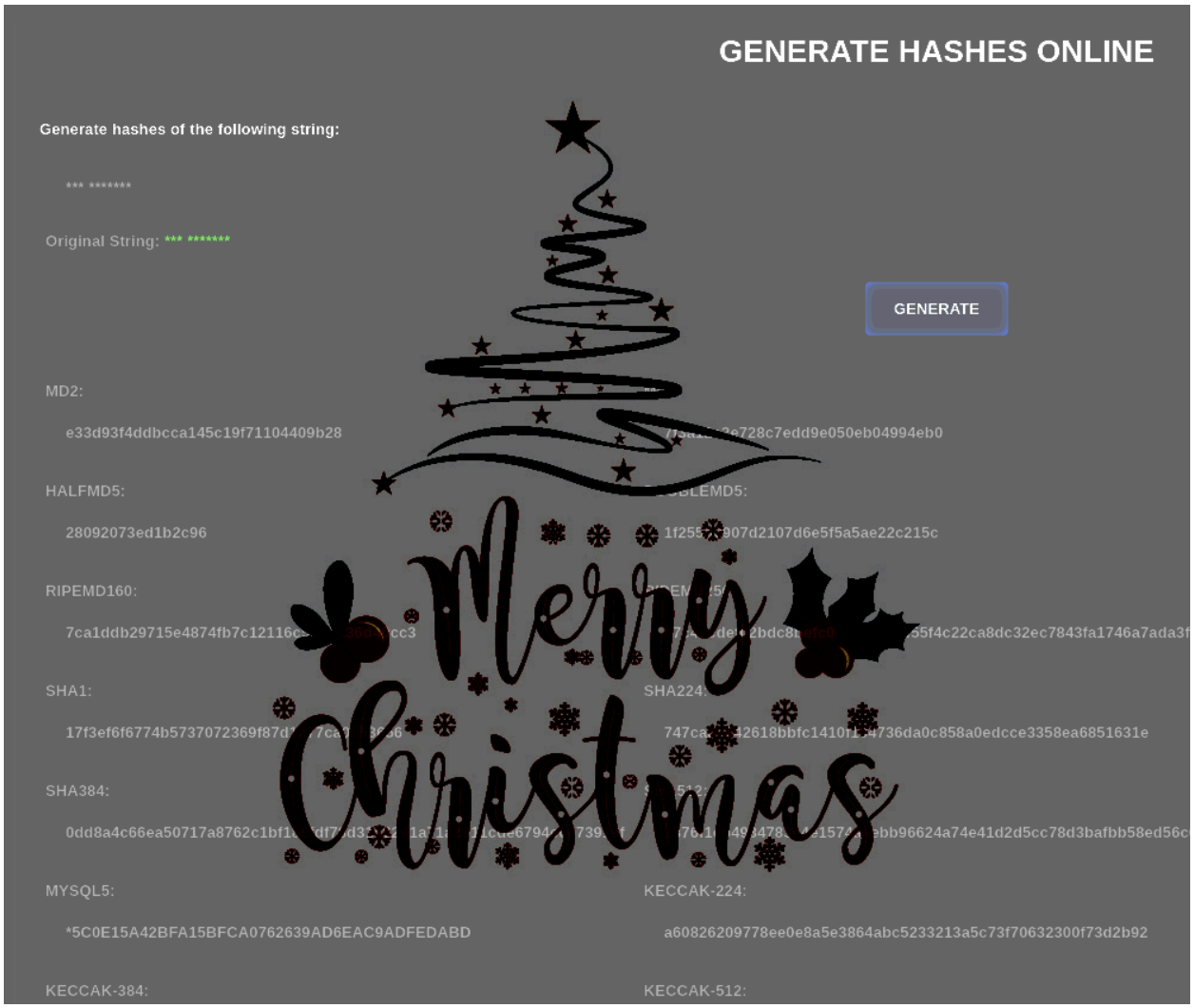
```
a32b698309d002df805f020c760ca997
```

The **part_2/part_2.zip** file appeared to be an encrypted zip file. The hashes were extracted with the tool **zip2john** and truncated into the hashcat format. Only one hash was required as they all had the same password.

./zip2john part_2/part_2.zip

```
$zip2$*0*3*0*7cd12c809de7025b5022ddef86a18344*b5b0*276*ddfa01ec956c3a
df3b8883f35542d9027f5cd845a2c8535b8e3873c0db2436aeee9dd6049bd45d2b196
f19c73fe891280291e77a0f9369e08366304b49e3c12997d7e20e05a22c46ce1f0a6c
aa29bbcc3181c8da2eef1512021d18c2278e9c0f7abdab6ccfe01af7cdc7a4923071e
c06e1df06dd4fe862cb80768bf468af15afab3a82de74a4d165ca89c80bc9f23ab507
05ab16149eab5eb1beb42aad918c80492ea134822a5260f678476e90ce96355f9871a
992e8a0ee0bc3b9bf94b7245c8fa31d94e665faca88c1b7e2cca83852304a758327a7
f061ebc620971bd4cfd73c72b960139929f29557c9e66db820483a39574398623cba7
879e377511a179b5408694e52407a80d241da875a962cb224e80adaddf5acb42a6404
1c2781a9e80e5da5296c7b9d64b8ff2501d6c9d239101c4cef56aa19f6ff6ea9c8bab
2896e2fd95216abde9bcfdfb8c03b8cab31152f659d1504cefaa9152b06faf4fe8bb1
e4c943992409b34719da67b7db2fde2a55ed26a8e2263b82d686ca883bae75103767a
a508c35da8d0e587de7254c0b5aabf202e6e7adab6c0b5203db308acb92d467b6d215
b3bfbcdb70ec54e9946ec325458015c5d29e8039a1d8720c8d23603e579f5d858b326
e497c46912a3cb56a3d5c61146e862a41609a4c1ba0b92a6ea70ccc05f67b6c6eba6b
c1d81990fd988799137a8e6681a90aeb4f1cd069429ddfe14efc26f40890b164574b5
df0e085224e620d3e438301d4f7a05d78ffeed6fa89b3f34a418d2c96380b483da576
ac9babbe241f161a99ff44e21e7928e8b072e37a4ed53aa6d02bd1731000802f781fa
9c8e9b6a1e8bea989aabb8ad61855737b37ac4c0da0cd8c2d159afe142197e15b4c31
9084*f31cad82ca78fd50b734*$/zip2$
```

The **merry_christmas.png** contained an image which clearly had text semi hidden in the background. After adjusting levels it became clear that this was a number of different hashes generated utilizing a hashes.com tool (https://hashes.com/en/generate/hash)



It was quickly discovered that the hashes were all cracking to the string “red herring”, leading us to believe it was a dead end. However, we thought some combination of fronts and backs of these hashes could possibly be the answer. We tried this extensively on both the zip hash and the hint hash with both hashcat and mdxfind, thinking perhaps the **hint.txt** hash was an alternate algorithm or perhaps even a custom number of iterations. This failed and we went back to the riddle and analyzed the PNG file.

With various steganography tools, no hidden data was discovered in the PNG file. We also generated MD5 checksums of the 3 files and used that data, but we still had a missing fourth part. After clues were posted to the forum, we considered every possible source of MD5 hashes, including the literal MD5 hash in the PNG image (of “red herring”).

The riddle seemed to indicate that both front and back parts of the hashes could be used. Thus we assumed equal length segments of 8-characters from the beginning and end of hashes.

54ad94b17c0292bd9c1db589**4795d718** - merry_christmas.png
d9598da6a25b759cdf6013ee**aca406c7** - riddle.txt
446d0265702ecb51a248337a**aa016ba2** - hint.txt
28092073ed1b2c9697e79ac8**68175964** - "red herring"

Utilizing a custom combinator script (that relies on combinatorX from https://github.com/hashcat/hashcat-utils) we ran through the strings in bold above and successfully cracked the part_2 zip hash.

./combine parts | hashcat -m13600 part_2.hash

```
$zip2$*0*3*0*7cd12c809de7025b5022ddef86a18344*b5b0*276*ddfa01ec956c3a
df3b8883f35542d9027f5cd845a2c8535b8e3873c0db2436aeee9dd6049bd45d2b196
f19c73fe891280291e77a0f9369e08366304b49e3c12997d7e20e05a22c46ce1f0a6c
aa29bbcc3181c8da2eef1512021d18c2278e9c0f7abdab6ccfe01af7cdc7a4923071e
c06e1df06dd4fe862cb80768bf468af15afab3a82de74a4d165ca89c80bc9f23ab507
05ab16149eab5eb1beb42aad918c80492ea134822a5260f678476e90ce96355f9871a
992e8a0ee0bc3b9bf94b7245c8fa31d94e665faca88c1b7e2cca83852304a758327a7
f061ebc620971bd4cfd73c72b960139929f29557c9e66db820483a39574398623cba7
879e377511a179b5408694e52407a80d241da875a962cb224e80adaddf5acb42a6404
1c2781a9e80e5da5296c7b9d64b8ff2501d6c9d239101c4cef56aa19f6ff6ea9c8bab
2896e2fd95216abde9bcfdfb8c03b8cab31152f659d1504cefaa9152b06faf4fe8bb1
e4c943992409b34719da67b7db2fde2a55ed26a8e2263b82d686ca883bae75103767a
a508c35da8d0e587de7254c0b5aabf202e6e7adab6c0b5203db308acb92d467b6d215
b3bfbcdb70ec54e9946ec325458015c5d29e8039a1d8720c8d23603e579f5d858b326
e497c46912a3cb56a3d5c61146e862a41609a4c1ba0b92a6ea70ccc05f67b6c6eba6b
c1d81990fd988799137a8e6681a90aeb4f1cd069429ddfe14efc26f40890b164574b5
df0e085224e620d3e438301d4f7a05d78ffeed6fa89b3f34a418d2c96380b483da576
ac9babbe241f161a99ff44e21e7928e8b072e37a4ed53aa6d02bd1731000802f781fa
9c8e9b6a1e8bea989aabb8ad61855737b37ac4c0da0cd8c2d159afe142197e15b4c31
9084*f31cad82ca78fd50b734*$/zip2$:d9598da6aa016ba24795d71868175964
```

After Hashes revealed the importance of the hint with the emoticon on the forum, we were able to crack **hint.txt** by combinating various UTF8 image characters.

A32b698309d002df805f020c760ca997:➡️⬅️⬅️⬅️

We believe in hindsight this plaintext was intended to hint at three sections coming from the end of the MD5s, and the fourth section coming from the start of an MD5.

Part 2

We found that part_2 was extremely difficult and did not solve it until Hashes posted extensive hints. Three files were extracted from the zip file:

```
riddle.txt
happy_holidays.png
part_3/part_3.zip
```

The **riddle.txt** file contained:

```
Beyond the bright display you see,
A hidden record holds the key.
In cryptic corners of creation’s code,
Clues lie waiting, quietly bestowed.
Seek the silent story that words can’t tell,
And a festive secret shall reveal itself well.
```

The riddle seemed to indicate that data was hidden inside the image. Indeed we found text both literally inside the image with levels adjusted, and hints inside the metadata of the file itself.

Steganography analysis was performed on the PNG file with various tools, and it immediately became clear that data was hidden inside the file:

exiftool happy_holidays.png

```
Comment : 🧑‍🍳 Password: d79f7c4569781c32d56f92c51814bd97  \ 🎄Hint:
68f0c603e9ad836c80e071b467a0f8d6
```

After many hours of trying various ideas to crack these hashes, we were effectively at a dead end. Hashes then posted numerous hints including the plain for the **part_3.zip** itself as there appeared to be an issue with this section. The plain was “merryhashmas2024” which was a combination of terms used in the filenames themselves, including the original zip archive and the current year.

Part 3

The **part_3.zip** file expanded into two files:

```
riddle.txt
hashes.txt
```

The **riddle.txt** file contained:

```
Shadows split into eights and twelves,
Secret sums hidden on the shelves.
A stray letter stands beyond the usual line,
G among the hex—a puzzling sign.
Combine the fragments, reveal what they hide,
And from hashed illusions, the key shall reside.
```

The **hashes.txt** file contained 10x hashes with a length of 32-characters, that turned out to be MD5s:

```
f4e56e399640b1b8645f7ddb58ca782f
59d054db88992a4ed1bbe2efe61f9d78
03fb5d283764ce52f73332160e3b37e3
52c471fe5a0f9c78bd460d349d0dc0ec
8f2f3f9d3b1a21b4ac251c68403e95b7
38ada4566dde81062e827340ffb1e96b
fbd47d503a7d13284582b6c31e618c74
9ae9512140703964b7702788dfdf2a2c
13ff27f5b526502cb56f94b0fb9cef5d
039f9b1a13c2ffb53b04a580dae223a4
```

After failing to crack these hashes with various wordlist and combinator attacks, we guessed they possibly could be hex and represent some kind of crypto private key. Thus brute force attacks were run for both `?h` and `?H`, up to 12 positions as specified in the riddle. This cracked 8 hashes.

`hashcat -m0 -O hashes.txt -a3 ?h?h?h?h?h?h?h?h?h?h?h?h --increment`

`(failed)`

`hashcat -m0 -O hashes.txt -a3 ?H?H?H?H?H?H?H?H?H?H?H?H --increment`

```
9ae9512140703964b7702788dfdf2a2c:02B1F6CE
fbd47d503a7d13284582b6c31e618c74:CE6BF76E
13ff27f5b526502cb56f94b0fb9cef5d:9122CF99
f4e56e399640b1b8645f7ddb58ca782f:1631DF8A3A7B
52c471fe5a0f9c78bd460d349d0dc0ec:B7E4C671
039f9b1a13c2ffb53b04a580dae223a4:5FE1A5D2
59d054db88992a4ed1bbe2efe61f9d78:5A7C819E6786
8f2f3f9d3b1a21b4ac251c68403e95b7:C6EBF7E61B6F
```

As two hashes were left, we went back to the riddle, which also mentioned that a “stray letter” was present, leading us to test adding “G” to the mask attack. This cracked the 9th hash.

`hashcat -m0 -O hashes.txt -1 ?HG -a3 ?1?1?1?1?1?1?1?1?1?1?1?1 --increment`

`38ada4566dde81062e827340ffb1e96b:F66GC671C86E`

The 10th hash was more difficult to crack, as we initially assumed it would actually be a full crypto private key of some sort and tried lengths such as 64 characters. Eventually we tried combinating all sections of the plains already cracked which led to the 10th hash being solved:

`03fb5d283764ce52f73332160e3b37e3:F666C671F66GC671C86E`

When we analyzed this plaintext, it became clear that it was an apparent clue from the designers:

`F666C671 F66GC671 C86E`

The first two substrings are nearly identical, save for a “6” in the first string, and a “G” in the second string. As “G” is not a valid hexadecimal character, we assumed it should be replaced with the “6” in the previously cracked hash 9. The last four characters being present was curious… but we were not initially sure what to make of it.

Having cracked all 10 hashes, we knew some kind of crypto private key, seed phrase, or otherwise was likely hidden in the plains. As they were already in hexadecimal, we thought combinating them in various fashions may lead to the right key.

We referenced the length of a SHA256 Bitcoin private key, which is 64-length hex, then combinated the 8 and 12 length strings from the cracked hashes in various fashions to produce 64-length hex strings. It should be noted we indeed replaced the “G” with a “6” in the 9th hash, and completely left out the 10th hash as that seemed to be a clue, not actual data. Here is what we initially combinated:

```
02B1F6CE
CE6BF76E
9122CF99
1631DF8A3A7B
B7E4C671
5FE1A5D2
5A7C819E6786
C6EBF7E61B6F
F666C671C86E
```

```
./combine plains | grep -Pa '^.{64}$' > generated_keys.txt
```

The generated keys would have to be converted into Bitcoin addresses, which was performed with bitcoin-tool (https://github.com/matja/bitcoin-tool). We generated both compressed and uncompressed addresses, in the formats base58check and bech32:

```
./bitcoin-tool --network bitcoin --input-type private-key
--input-format hex --output-type all --output-format base58check
--public-key-compression compressed --batch --input-file
generated_keys.txt | grep -Pa 'address-checksum.base58|address.bech32'
| cut -d: -f2- > generated_addresses.txt
```

We also needed a method to check Bitcoin address balances. Two solutions were developed, a custom Python tool that generated all possible 64-length hex private keys from the above wordlist and checked them for balances with an API, and utilizing an offline database (the Blockchair Bitcoin Addresses dataset) which only includes addresses with balances (https://blockchair.com/dumps).

After the Blockchair database was acquired and addresses parsed out, we searched all funded BTC addresses for our generated addresses using a hash search method in a Perl script.

```
#!/usr/bin/perl

use strict;
use warnings;

my %hash;

open(FUNDED_ADDR, <$ARGV[0]>);
open(GENERATED_ADDR, <$ARGV[1]>);

while(my $line = <GENERATED_ADDR>) {
  Chomp $line;
  $hash{$line} = 1;
}

while(my $line = <FUNDED_ADDR>) {
  chomp $line;
  print $line . "\n" if $hash{$line};
}
```

This script was run:

./findhash funded_addresses.txt generated_addresses.txt
(no output)

This process failed however, and no generated addresses matched funded addresses. We again looked at the riddle and analyzed the plains cracked. While testing various ideas, we went back to how it was curious that `C86E` had been present at the end of the 10th hash “hint”.

It seemed like maybe that part of the plaintext should also be replaced? A new combinator attack was started with `C86E` being removed from the modified 9th hash (where we replaced the “G” with the “6”. This made the string 8-characters instead of 12.

`F666C671C86E -> F666C671`

With our new list we again combinated, converted to BTC addresses, and compared against the Blockchair funded addresses list. The combinator wordlist was now:

```
02B1F6CE
CE6BF76E
9122CF99
1631DF8A3A7B
B7E4C671
5FE1A5D2
5A7C819E6786
C6EBF7E61B6F
F666C671
```

This time we got a match on an address!

```
./findhash funded_addresses.txt generated_addresses.txt
1GWGfS5WpeKdhvai2VEtJVrogxUeLbhtfU
```

We quickly checked the “Blockchair” data for this address and saw that it contained the exact amount Hashes had mentioned on the forum. Additionally, blockchain.com showed the address was funded on Christmas Day. It seemed that we had the right wallet!

Wallet data including the private key was hastily generated utilizing bitcoin-tool:

```
./bitcoin-tool --network bitcoin --input-type private-key
--input-format hex --output-type all --output-format base58check
--public-key-compression compressed --batch --input-file
generated_keys.txt > bitcoin_data.txt

grep -A50 -Fa 1GWGfS5WpeKdhvai2VEtJVrogxUeLbhtfU bitcoin_data.txt

address.base58check:1GWGfS5WpeKdhvai2VEtJVrogxUeLbhtfU
address.bech32:bc1q4gfsvtk80kqsm7htk6ztsvul0rzx50as6s5mhu
…
private-key-wif.base58check:
KwxrXUykANd2dJKpNPZWJMayTH5xCmsbXdF7x2DzjFdm261JixHQ
```

Note that the actual private key in hex showing the distinct combinated parts was:

```
1631DF8A3A7B 5A7C819E6786 F666C671 CE6BF76E 02B1F6CE 9122CF99 5FE1A5D2
```

Now having the base58check private key, we were able to sweep the funds from the wallet.