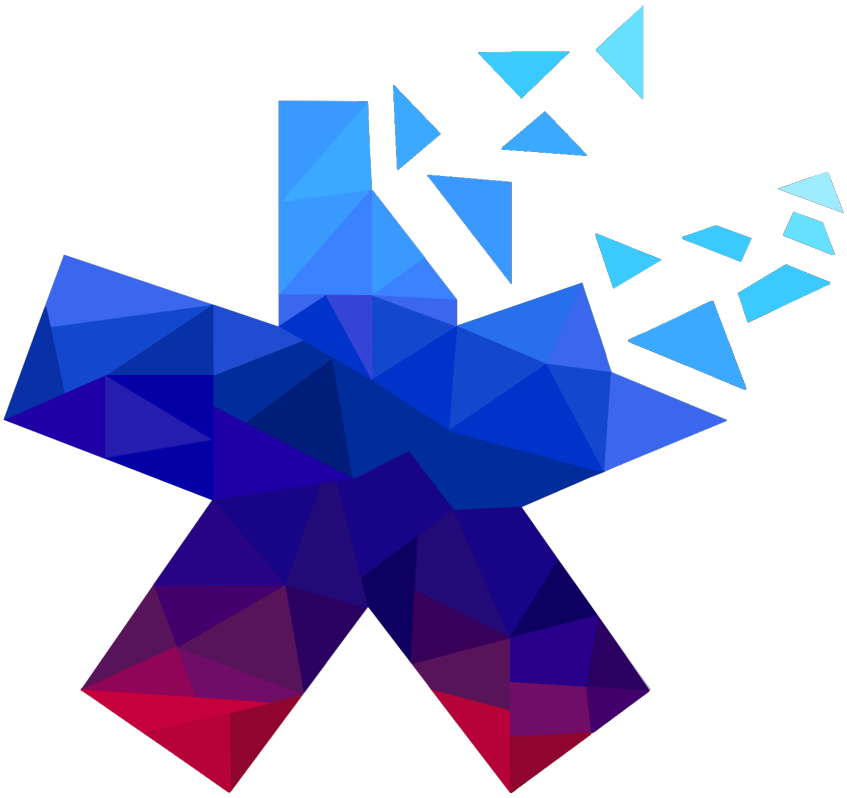
HashMob

Historic. A rollercoaster of emotions and a lot of pastries. I ate a lot of pastries

---

Paris, 2024

Phil Wizard



CrackTheCon 2025 Write-up

HashMob.net
vavaldi@hashmob.net

Chapter 1

Preface

Another year for an awesome contest. We're very happy that the wonderful members of CynoSure Prime (CSP) decided to push through with hosting the contest. Even if time allocations were limited, we were glad your work came to fruition in what can only be described as the most nerve-wracking 24-hour photo finish we've had (thus far). HashMob participated with a single street team during the contest due to the 'closing' of the pro-division and merge with the street division (a decision made due to lack of sign-ups). This marks our second CTC victory and we are very happy to have come out on top after such a long and intense battle. Huge kudos to Unmuddlers (which included a smaller subset of Team Hashcat) for keeping us on our toes.

Unfortunately the contest was hosted on week-days meaning that a large part of the team could either not participate or were limited in their participation.

The Team

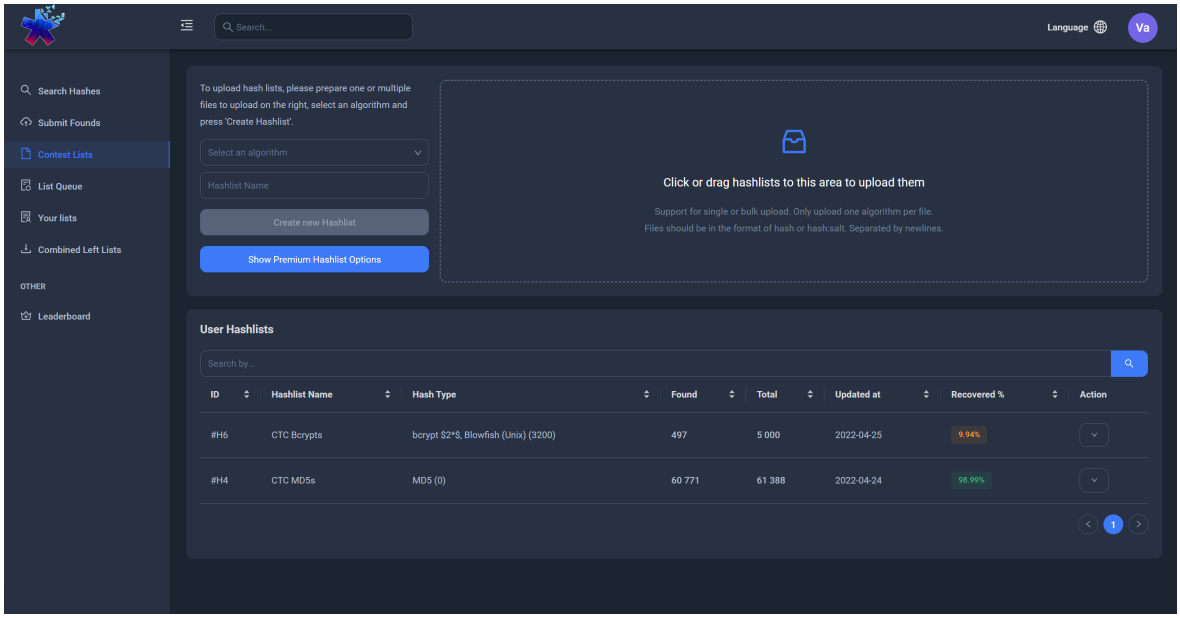
- Vavaldi
- penguinkeeper
- Shooter3k
- _cin
- justpretending
- _0.0.0.0_
- afsa
- AdamBlack
- TalentedGuy
- cyclone
- Brad
- Cake
- mostwanted002
- segment
- SoSander
- kpd

1.1 About HashMob

HashMob is a large, mostly Discord based, community which focuses on Cryptography and Hash password recovery. Users have picked password recovery up as a hobby over the years due to their interest in security or because of their jobs, and spend a lot of time working with cracking hashes and research on the passwords of users. It was founded in 2021 almost half a year after Hashes.org closed its doors in January of 2021. Since then it has recovered over 559 919 341 new passwords / plaintexts on top of Hashes.org and amassed a following of more than 5,000 members.

1.2 Contest Environments

We created a copy of the HashMob.net web application with a custom back-end script that submit new solutions automatically to the CrackTheCon API-endpoint. These environments were restricted in access, so only authenticated team members could participate. Additionally, we had set up several custom applications and tools such as Hashtopolis, a custom sync script to share founds via the HashMob copy, and hashcat builds to automatically set parameters based on HashMob IDs as well as some other features and unmerged hashcat GitHub PR's.



Chapter 2

HashMob write-up

2.1 The Preparation

In preparation of the contest, we tried recovering the example hashes until we found out that HashMob had all hashes pre-cracked already.

2.2 Software Used

Listed below are some of the software used by the team, although a majority of them are public / open source tools, some are closed source, modified open source, or specifically developed for the contest. Unfortunately, a large part of the complex infrastructure we set up in 2024 for CMIYC was lost due to a triple-disk failure which affected HashMob and its surrounding environments. Unfortunately it's hard to predict so many failures in such a short time (and yes, we did use RAID and a 3-2-1 backup solution which is why we were able to recover almost everything). Lessons were learned; and now we can handle 3 disk failures for nearly everything. *knocks wood*

The following table presents an overview of some of our core tools. This is not always an exhaustive list as we might smash together scripts or use external software based on things thrown at us. Anything noteworthy will be listed here however.

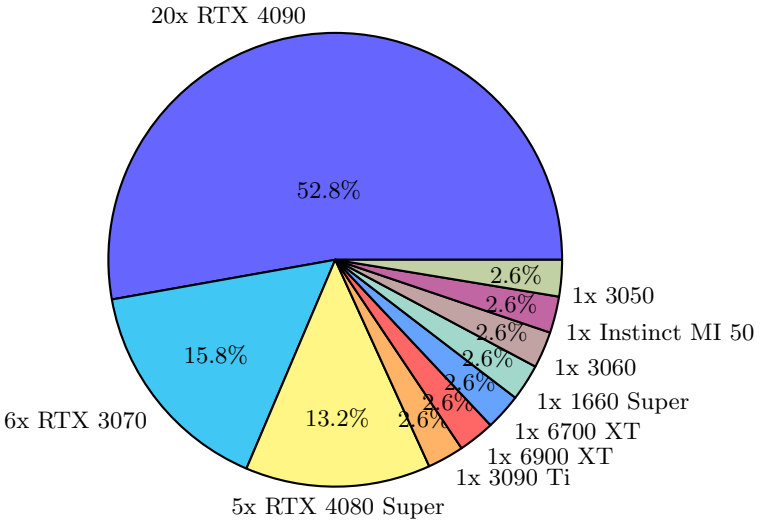
Overview of Used Software			
Name	**Open Source**	**Public**	**Purpose**
autocrack	no	no	Auto Cracking
bkcrack	yes	yes	Zipcrypto recovery
debug_rule_submitter	no	no	Auto submission
debug_rule_receiver	no	no	Auto submission
Hashcat*	yes**	no	Password recovery
MDXfind	no	yes	Password recovery
hash_finder	no	no	Analytics
hashgen	yes	yes	Hash generation
HashMob Search*	no	yes	Hash Lookup
Hashtopolis	yes**	yes	Collaboration
HashMob Mirror	no	no	Collaboration
gocrack	yes	yes	Hash generation
Gramify	yes	yes	Analytics
JohnTheRipper	yes	yes	Password recovery
PACK	yes	yes	Analytics
PACK2	yes	yes	Analytics
plain_finder	no	no	Analytics
ptt	yes	yes	Analytics
RuleProcessorY	yes	yes	PW generation
PRINCE	yes	yes	PW generation
PCFG_Cracker	yes	yes	PW generation
Spider	yes	yes	Scraper
sync.py	no	no	Auto submission

* These tools can (also) be found on HashMob.net or their Discord.

** Source code was modified and tweaked to suit our needs.

2.3 The Hardware

We've been getting more comfortable with using rented hardware to help deal with larger workloads or run more complex attacks - trying to optimize attacks sometimes takes more time than is worth to just throw a little more power at it. But CTC and CMIYC are contests very much designed to prioritize thinking and efficiency over raw power and though we did win the doubling of power would not equal to a significant increase in cracks. A total of 36 Graphics cards were used from our own physical machines. Additionally, we also rented: 16x 4090 for 2 hours, 2x 5090 for 15 hours, 4x 4090 for 70% of the contest, 4x MI300X for final 30 minutes, 14x 4090 for a part of the contest. We might have missed some in this overview as we didn't receive a response from everyone - but does largely cover our utilization.



2.4 Ready? Set...

Go! After having collaborated together numerous times we have gotten a better feel for the start of the contest and working together. Compared to earlier contests you can definitely notice a difference. We better distributed the workload in creating Discord channels (which we use as main form of communication), creating hashtopolis hashlists, and getting everyone working on different lists so that we would be able to cover more ground.

2.5 The Contest & Results

The lists presented to us for the 2025 CrackTheCon contest were made up of various algorithms. We'll go over each hashlist one at a time, discussing how we approached it, what we found and the final conclusion. If you are interested in taking on the 2025 lists yourself, we recommend stopping after this section. The lists we were offered were named: bcryptsalot, cewlcon, SaltyHashAlgo1, SaltyHashAlgo3.1, SaltyHashAlgo3.2, touchgrass, zipidydoda, WTF_Rob, Cyberpunk, tech_support, and finally Quorum_Quest.

HashMob finished in first place with a total of 141 998 670 points, extremely closely followed by Unmuddlers with 141 869 240 points. First blood points gained an extra 10%.

Hashlist	Plains	First Blood	Point Value	Score
Cyberpunk	295 306	203 613	200	63 133 460
bcryptsalot	4 966	4 709	4 000	21 747 600
touchgrass	95 244	11 195	200	19 272 700
SHashAlgo3.2	3 399	2 213	1 200	4 344 360
tech_support	38 555	882	800	3 154 560
cewlcon	14 856	9 495	100	1 580 550
SHashAlgo1	463	375	2 000	1 001 000
Zipitydoda	263	263	800	231 440
Quorum_Quest	74	74	2 000	162 800
WTF_Rob	105	57	1 000	110 700
SHashAlgo3.1	0	0	1 100	0

Total Ranking

#	Team Name	Founds	First Founds	Points
1	HashMob.net	453'231	232'876	141'998'670
2	Unmuddlers	452'794	149'867	141'869'240
3	UNSHADE	423'986	11'335	107'857'590
4	Pirates247	280'355	11	77'873'120
5	hash_meltdown	227'096	75'398	46'341'420
6	hashmouse	83'472	317	16'073'020
7	OwMyElectricityBill	59'161	4	13'945'740
8	CANB0$$	44'969	9	8'624'080
9	Blugold Group	24'922	213	4'957'580
10	pigswny	13'973	0	2'821'800
11	chi_cracked	13'718	0	2'745'000
12	P@ssw0rd	9'561	10	1'164'320
13	Sigpwny	175	0	35'700
14	Shelly	0	0	0
15	soup	0	0	0
16	PrincessTG	0	0	0
17	Xorcrypt	0	0	0
18	I have no friends	0	0	0

2.6 Quorum_Quest

Hashlist	Plains Found	First Blood	Score
Quorum_Quest	74	74	162 800

V: We're starting this writeup with one of the more technically interesting ones as I feel it'll help grab your attention and make you read the rest as well. The rest is sorted by score, Enjoy!

We were given the 3 archives p1.rar, p2.7z and p3.arc, and a text file called sss.txt. The text file contained some hex data:

```
804333f9443c0a90b39be2f322ca828497be97fc9b37342626762edcc467ea6eb1ba43c31a6e11
459c1275712dc18dbd2e2ec86d924c8f84038ab4a719ee0af185e67a9a2df5e573aaabf2142980
e2968a7a6dd8d7e6287076ad666cd87c3747d15e93a1fd227d81532eb9d5c25a24519f5a8c4155
fd7d233314e117ed3669734e3c6ffa390624e3a83dee8070d3e6a2b7b46499c7dc231c72939ce8
ca6e2a99a0bd076f642388f97bbfc0172fc58ad61b625264c26ff86a44b02ea719877a0a8444d0
db4f893083289b425af7c1dc543404
```

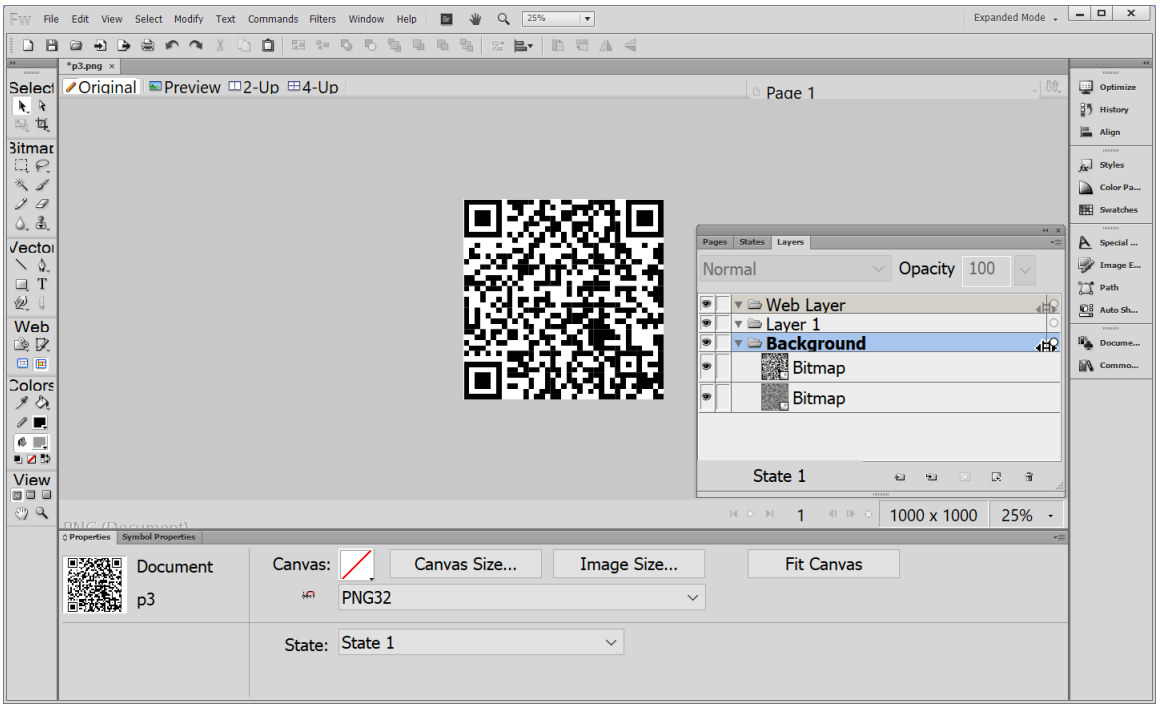
The passwords for "p1.rar" and "p2.7z" were quickly cracked with a simple dictionary attack for which the passwords were "secret" and "sharing" respectively. Both archives contained a PNG file of a QR code containing hex data similar to "sss.txt".

The passwords, together with the name of the text file "sss", led us to believe that Shamir's Secret Sharing (SSS) algorithm was being used, an algorithm by which you can split data into N segments but only need a portion of those to recover the original data. This - by concept - allows you to split a secret between 10 people but only require any 5 to recover the secret. This meant that we had to likely also crack the third archive as well to reveal the secret.

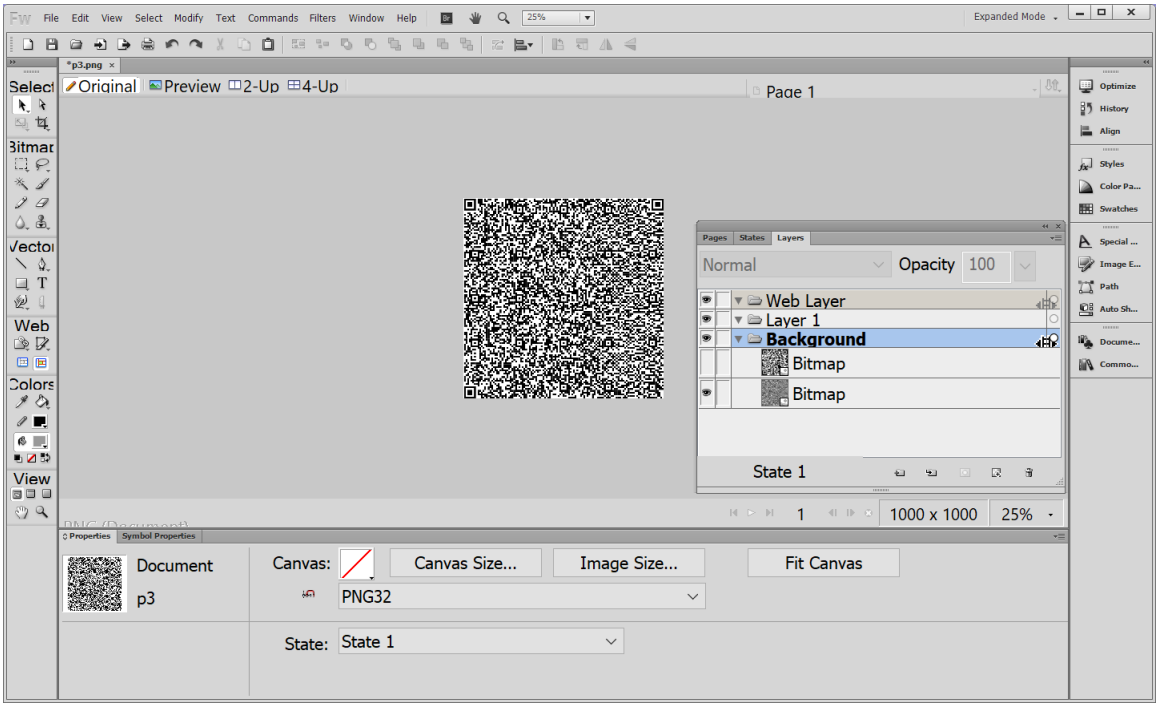
Given the passwords for the other two archives, it was very likely that the password for the ARC archive was "shamir's". After installing FreeArc, this quickly proved to be the case. However, the PNG file inside the archive contained a QR code that linked to a "Rick Roll" YouTube video. After some struggling and unsuccessful attempts with various steganography tools and methods to extract the real data we needed, we took a closer look at the metadata of the PNG file (next page):

```
afsa@kali:~$ exiftool Downloads/p3.png
ExifTool Version Number         : 13.10
File Name                       : p3.png
Directory                       : Downloads
File Size                       : 240 kB
File Modification Date/Time     : 2025:04:01 16:23:35+02:00
File Access Date/Time           : 2025:04:07 11:01:26+02:00
File Inode Change Date/Time     : 2025:04:03 11:11:08+02:00
File Permissions                : -rw-------
File Type                       : PNG
File Type Extension             : png
MIME Type                       : image/png
Image Width                     : 1000
Image Height                    : 1000
Bit Depth                       : 8
Color Type                      : RGB with Alpha
Compression                     : Deflate/Inflate
Filter                          : Adaptive
Interlace                       : Noninterlaced
Significant Bits                : 8 8 8 8
Pixels Per Unit X               : 2834
Pixels Per Unit Y               : 2834
Pixel Units                     : meters
Software                        : Adobe Fireworks CS5
XMP Toolkit                     : Adobe XMP Core 5.0-c060 61.134777, 2010/02/12-17:32:00
Creator Tool                    : Adobe Fireworks CS5 11.0.1.7 Windows
Create Date                     : 2025:04:01 14:23:13Z
Modify Date                     : 2025:04:01 14:23:35Z
Format                          : image/png
Image Size                      : 1000×1000
Megapixels                      : 1.0
```

Of particular interest was the software that was used to create the file: Adobe Fireworks CS5. This tool wasn't used to create the other two images. Googling for possible uses of Adobe Fireworks to hide data in PNG files led us to the following StackExchange question: https://puzzling.stackexchange.com/questions/63626/hidden-message-embedded-in-image It turned out that the same thing was done with the PNG file we received. Opening the file in Adobe Fireworks showed that it also contained two layers (next page):

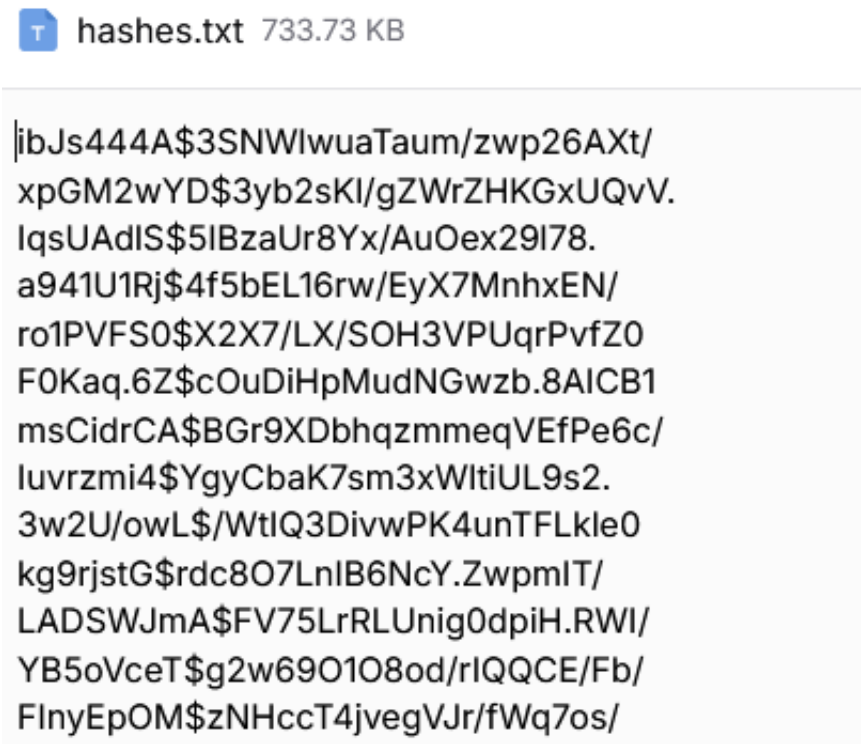


The second layer contained the cleverly hidden QR code we actually needed:



The QR code also contained hex data, just like the other two. With that we had all the parts we needed. The last step was to find out which tool could be used to combine the parts to reveal the secret. A closer look at the 4 hex strings showed that they started with 801, 802, 803 and 804 respectively. The same prefixes are also used by this tool: https://iancoleman.io/shamir/

Combining all 4 parts in the tool by iancoleman gave us a link to a file share where we could download the actual hash list:



We quickly identified these hashes as Apache $apr1$ MD5 hashes with the "$apr1$" prefix missing and resulted in several cracks of both n-grams and heavily mangled passwords. We noticed similarities with other lists as well when it came to the plains themselves. Identifying plains that overlapped partially or fully indicating they might be originating from a similar source. Unfortunately we were unable to recover a lot of plains.

```
$apr1$UYm0nqSV$2dyVjJL3HSVcqSyN0NHqt.:believed she
$apr1$KTDDJN1Q$n0.ef0j2YKpjJqxujlKeV.:associated with
$apr1$4rtUqfWY$mXBbewTgMCXARhjKMMFnR.:minutes,
$apr1$jC9YnysH$wkeN/eEWp.dDJA/S6aWD6/:that didn't
$apr1$C9.mMzha$U97SxBfn2luQysvwU.qT/1:the ground
$apr1$q/EospA7$MvQQvT766MxpVhU/5gYUO/:reaches of
```

2.7 Cyberpunk

Hashlist	Plains Found	First Blood	Score
Cyberpunk	295 306	203 613	63 133 460

This hashlist contained a single file - "Cyberpunk.dmp" which the 'file' command identified as being an ELF file. After searching through the file for readable strings, we discovered that this was a dump of a Windows VM, generated in VirtualBox.

With this information, we searched for tools that were capable of reading such files, with Volatility3 being perfect for what we wanted. Using this tool, we found the functionality to dump the user hashes straight from the SAM using a plugin called "windows.registry.hashdump.Hashdump" - something that looked very appealing for a hash-based challenge. After running this for a while, Volatility dumped around 300k Windows account records, containing usernames and NTLM hashes.

```
william.collins 1068    aad3b435b51404eeaad3b435b51404ee        7e2be013b0ef0586a9966f7cb77def0e
william.reyes   1069    aad3b435b51404eeaad3b435b51404ee        f611c0c1a4e6f583cf7ff052dd45cf0a
william.stewart 1070    aad3b435b51404eeaad3b435b51404ee        3eb8e2c7e78742e696648e4bb5ac162a
william.morris  1071    aad3b435b51404eeaad3b435b51404ee        945b9ee858bc43ba105851dfe9239a5f
william.morales 1072    aad3b435b51404eeaad3b435b51404ee        42e05f73417eb4028dedcbb20889b6ba
william.murphy  1073    aad3b435b51404eeaad3b435b51404ee        4fe7252f91b52691b7aa9c203af576e2
william.cook    1074    aad3b435b51404eeaad3b435b51404ee        004c624e3ba309ab47985caea71c1124
william.rogers  1075    aad3b435b51404eeaad3b435b51404ee        16f30b625b545b2ed1e437ad0e5ce747
william.gutierrez       1076    aad3b435b51404eeaad3b435b51404ee        61b50cfad3910a749643ea1e3402cd1b
william.ortiz   1077    aad3b435b51404eeaad3b435b51404ee        1b493f60314c71cf6635dbda20f3752c
william.morgan  1078    aad3b435b51404eeaad3b435b51404ee        d4556fb1c915d9d7f49a6bbffb08ab2a
william.cooper  1079    aad3b435b51404eeaad3b435b51404ee        56f598bce39fd9604320de7c5a6efff0
william.peterson        1080    aad3b435b51404eeaad3b435b51404ee        00bbab35b634bbbaf5431be04edd2afb
william.bailey  1081    aad3b435b51404eeaad3b435b51404ee        71e407115b511344a7432ea48979b85d
william.reed    1082    aad3b435b51404eeaad3b435b51404ee        23255d0b973d0894e697e183fcf73449
william.kelly   1083    aad3b435b51404eeaad3b435b51404ee        63a08c33557eb1c9682baac230e417d4
william.howard  1084    aad3b435b51404eeaad3b435b51404ee        af94d47759babca062b8c2d3914befa4
william.ramos   1085    aad3b435b51404eeaad3b435b51404ee        60ff930b619777d625a4e6baeb588d32
```

Running some basic attacks against the NTLM hashes we discovered a mix of normal-looking passwords, and passwords themed after the video game "Cyberpunk 2077"- matching the name of the file, "Cyberpunk.dmp". Many of the initial founds hinted at the Cyberpunk series and included themed words and sources. Many founds were identified to be using the same base word, with many mutations applied using rules. Using debug data, the team was able to identify top-performing rules, then seek out key n-grams for obtaining more founds.

We found that scraping Cyberpunk wiki's such as Fandom and Fextralife were very effective, especially when used with an n-gram generator such as Gramify in combination with a high amount of rules. Frequent patterns that emerged were character names, place names, items. and abilities which were then heavily distorted before being hashed.

We tried pairing the given usernames with other hashes found in the challenge, or using them as a wordlist itself, however this line of thought didn't get anywhere and we believe the usernames were simply randomly generated for flavor.

2.8 bcryptsalot

Hashlist	Plains Found	First Blood	Score
bcryptsalot	4 966	4 709	21 747 600

This hashlist was a tough nut to crack and a list we definitely could have hit harder. However, bcrypt is a very slow and expensive algorithm, and without clear optimized attack-path it is very challenging to get cracks. We utilized the founds verbatim from zipidydoda's CSV's to crack a large amount of bcrypt hashes. Other than that, we were unable to identify any significant patterns. We highly recommend reading the zipidydoda write-up.

2.9 touchgrass

Hashlist	Plains Found	First Blood	Score
touchgrass	95 244	11 195	19 272 700

We identified this list as SHA-1 after running several basic wordlist and rule attacks. ignis10M with top 500 rules got roughly 4,000 cracks. After inspecting and analyzing the cracked hashes we noticed a recurring theme about nature trails.

Rules were created to identify keywords from the found information. Top tokens were taken and nature themed words such as trail and cove were identified. These were transformed into targeted rules that appended trail or removed portions of the text before appending themed words. These processes revealed the list was heavily constructed using specific trail sources and should be scraped from online websites.

We used several sources to scrape trails, including fs.usda.gov, halfwayanywhere.com, trailforks.com, and alltrails.com. When using alltrails we used several methods of scraping. One was by using the robots.txt to locate the XML sitemap which contained a total of 550+ files. We unfortunately, quickly hit a rate-limit and several bans by the website likely due to the lack of JavaScript support, bad user-agent, or high amount of threads causing many requests to occur at once forcing an autoban.

Trails Directory: 'A' - Page 1

Browse by A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A Soustons, circuit à la découverte de la forêt ...	A Soustons, circuit à la découverte de la forêt ...	A - B - C Loop
Å - Stokkvika	A Aire-sur-l'Adour, circuit des vallons des Arrib...	A and C Trails
A Anea - Río Masma	A Arjuzanx, Circuit du tour du Lac	A Arjuzanx, la Piste du Lac
A Azur, Circuit du Pont du Loup	A Balatonfelvidék Kincsei	A Banos, Circuit des Belvédères
A Barcala - Cecebre vía Paseo Fluvial do río Me...	A Barrel of Laughs - Sawdust Trail - Cranbrook ...	A Bastennes, Circuit du Luy
A Belhade	A Bölcső-hegy legendáinak nyomában	A Bostens, Petit Circuit des 9 Fontaines
A Bougue, Circuit de Laglorieuse	A Bougue, Circuit de Mégnos	A Bougue, le Marsan en VTT
A Bouny	A Brea - Río Robra - Constante - Os Outarelos	A Break via Somerset Trail
A Bretagne de Marsan, Circuit de Laborde	A Brocas, l'Écobalade des Lagunes	A Brocas, les Traces d'une Occupation Ancien...
A caballo de las montañas	A Campa do Val - A Cruz de Outeiro - Pena dos ...	A Canal Trail
A Cañiza por carretera	A Canle - Cabo Touriñán - Punta de Sualba	A Capbreton, Circuit du Tuc des Neuf Églises
A Capbreton, Parcours Pédagogique du Tuc	Á Capela de Nossa Senhora da Luz pela rua do...	A Carballeira - Vilar - Camaforte
A Carcares-Sainte-Croix, Circuit de Sainte-Cr...	A Carcares-Sainte-Croix, Circuit du Chemin d...	A Carreira - Praia de Miño - Praia de Perbes
A Casanova - A Pedreira	A Cazeres-sur-l'Adour, Circuit de Lamensans	A Cazeres-sur-l'Adour, Circuit du Cap de la Har...
A Césc - A Cára - Anzone	A Cèsc - Capanna Alpe Sponda	A cheval sur les 3 communes
A Coruña - Bastiagueiro	A Coventry Way	A Coventry Way - Brinklow Walk
A Coventry Way: Wolston Circular	A Cseresznyés környékén	A Cubela - Mirador de Louxoá
A Cycling Tour of the London East End	A Davis Trail	A Dél-Vértes Szépségei
À descoberta da beleza da Costa da Arrábida ...	A deux pas de la nature	A Doazit - Circuit des 3 Clochers
A Duhort-Bachen, Circuit de la Lande	A Era Punto - Refugio de Farrigüelo - Punta de ...	A Ermida - Cornería

We ended up using a Burp intruder attack in the last 20 minutes before the end of the contest to enumerate all possible combinations of trails from https://www.alltrails.com/directory/trails/A/1 (see screenshot). Enumerating 35 pages per character of the alphabet; resulting in $26 * 35 = 910$ queries. We then exported them from Burpsuite, unticking the base64-encoded option,

and using regex to extract the title from the results a mere 15 minutes before the end of the contest.

Then came running the actual attacks. By the time the contest ended we were still running attacks on the scraped data. Combined with zipidydoda, it was the last burst of points to help solidify our first place.

2.10 SaltyHashAlgo3.2

Hashlist	Plains Found	First Blood	Score
SaltyHashAlgo3.2	3 399	2 213	4 344 360

We identified this algorithm as SHA3-512 which contained several n-grams or password-phrase inspired passwords.

Many of the themes were not strongly correlated during the contest period, but concatenated names, prepending "?" and "??", and selected full sentence phrases from online texts seemed to be strong performers against this list. IMDB datasets yielded the best results for us, combining first names and last names of actors or movie titles as well as facebook_firstnames + facebook_last names with -j "^?" in -a1.

```
?malcolmsinclair-
julIeengelbr
?jessiejohnso50
?corlanderfl
?seanbarnear
?sarahalshar17
?joshhelmant
?kendrickreyn05
johannakelse75
joano'gorman1
?holliehopki06
?devincromw06
nathanbrodr4
?nickhardri79
naomiebaue06
?jonasaldis00
??alandacol37
?johnchuzha42
?bobbiesele93
?rosamclarn92
??k.j.lawlor
?millyblunde74
```

2.11 Tech Support

Hashlist	Plains Found	First Blood	Total
Tech Support	38 555	882	3 153 560

```
We just got access to this company's customer database server and PBX.
They have been monitoring their tech support staff for some time.
I would hate that job so much, lol. Their employees sound dead inside.
I pulled some of the call_logs from the PBX they are so depressing
https://gofile.io/d/a0H7Bf
Dual Core had it right https://www.youtube.com/watch?v=rZHFoowQENQ
I managed to dump all the customer hashes from the database.
I cannot crack the passwords for their customer accounts though, must be
some @PWTooStrong level of stupid


out of character note:
I'm so sorry for this challenge it was a 3AM sleep-deprived idea and I
have not had time to verify everything is 100% accurate if there are
problems apologies -winxp
```

Tech support contained the REAME text file shown above. This carried a link to gofile.io which contained 508 WAV files. After downloading and manually listening to a few we concluded that each file contained a set of spoken words with name, email, and password. From some manual listening and testing we verified that the hashlist was made up of password phrases hashed with SHA512, with words being separated by several different separators.

We utilized AI, and Speech to text conversion software to convert the spoken words in the WAV files to text. We then utilized the extracted data with several rules and combinators to enumerate possible patterns. Some separators in-between words included "-,_^".

```
word word word word word
word-word-word-word-word
word^word^word^word^word
word_word_word_word_word
```

Enumerating with rules revealed additional patterns such as:

```
(word-word-word-word-word)
word-word-word-word-word!
word-word-word-word-word1
```

The founds, after going through some enumerating and different parsing methods resulted in a total of 207 unique words that combined in 1-5 different ways (multiplicative combination) + rule variations resulted in a near-full recovery of all hashes. The last word we found was "joule" (which speech-to-text incorrectly recognized as "jewel"), and combining it with the other words resulted in the final 959 remaining hashes. A perfect score.

2.12 Cewlcon

Hashlist	Plains Found	First Blood	Total
Cewlcon	14 856	9 495	1 580 550

The Cewlcon list was a nice little play on words - a portmanteau - referring to a mix between the tool "Cewl" and conference where the contest was hosted CypherCon. This also formed a hint to the way in which cracks could be obtained.

The list was quickly identified as MD5 by pre-cracks from our database and running basic attacks. From this we could utilize crawlers such as Cewl to construct a wordlist based on the CypherCon website. Creating combinations of words (in sequence) (also called n-grams) of words on the website.

By running the wordlist generated by the crawler through several rules showed a clear indication we were on the right path. For example, there were a number of passwords that contained cyphercon:

```
62554fbc020a8b9e19bbf76ed3599965:cyphercon?
2cdff339bd6a648057dd6cbf701b33c9:CYPHERCON?
3e6ad6d4a0711b2f5081e1df287c3694:Cyphercon?
a5cee8412f8c258f0f81f58f10073d27:cYPHERCON?
De7e35f73b072106d1b51f6904ba5519:cyphercon!
249ec85e78a3742c4194cd975875336d:CYPHERCON!
972bbb48c1de101d3a8996232bd7e2be:Cyphercon!
483971a19e6f459ae62444f252fc9f36:cYPHERCON!
```

The analysis of the passwords consisting of single words showed a heavy use of different capitalizations, which could be attacked with the following hashcat rules:

```
l
u
c
C
```

The passwords did not contain prefixes consisting of special characters or digits. Though there were some common suffixes:

```
?
!
1
123
2018 - 2025   (year suffixes)
```

In some cases up to 3-part combinations like "?2025!" were identified. Therefore, the "–loopback" flag provided a lot of value - as did multiple rule files. The passwords we found contained single words from the website, speaker names, talk titles, etc. We also found some passwords with other conference names in them, such as DEFCON and BSidesKC. However, it appeared that these names came only from speaker bios. Crawling other security conference websites did not yield any more cracks.

Quite a few passwords in this list were longer than 31 characters, so it was important to also run attacks without the optimized kernel.

2.13 SaltyHashAlgo1

Hashlist	Plains Found	First Blood	Total
SaltyHashAlgo1	463	375	1 001 000

This list was a SHA1 algorithm (-m 120) that consisted heavily of n-grams which were cut off at certain lengths or word boundary and frequently contained special symbols or l33tspeak within (likely also added with rules). Similar to the other "Salty hashlists: prepending "?" and "??" as well as phrase attacks seemed to perform better against the target list.

2.14 Zipitydoda

Hashlist	Plains Found	First Blood	Score
Zipitydoda	263	263	231 440

This hashlist consisted of a folder containing a README file and a folder named @crack-theconcloud. This was reminiscent of a series of popular stealer logs called Everlasting cloud in style. Unfortunately the folder contained nearly 5000 password-protected archives using 14-character randomly generated passwords (discovered later).

```
-rw-r--r-- 1 root root 4.0K Apr  2 06:34 10.0.110.197[CN]_browser_export.zip
-rw-r--r-- 1 root root 4.0K Apr  2 06:34 10.0.156.126[CN]_browser_export.zip
-rw-r--r-- 1 root root 4.1K Apr  2 06:34 10.0.163.203[JP]_browser_export.zip
-rw-r--r-- 1 root root 4.2K Apr  2 06:34 10.0.180.138[DE]_browser_export.zip
-rw-r--r-- 1 root root 4.2K Apr  2 06:34 10.0.201.1[AU]_browser_export.zip
-rw-r--r-- 1 root root 4.0K Apr  2 06:34 10.0.236.128[GB]_browser_export.zip
-rw-r--r-- 1 root root 4.2K Apr  2 06:34 10.0.88.76[DE]_browser_export.zip
-rw-r--r-- 1 root root 4.0K Apr  2 06:34 10.100.136.69[IN]_browser_export.zip
-rw-r--r-- 1 root root 4.1K Apr  2 06:34 10.100.165.208[IN]_browser_export.zip
```

Each of these ZIP archives contained a CSV file and a README file which was then compressed and archived with a password. .zip archives are known to use PKZip encryption (PKZIP/ZipCrypto), which is vulnerable to a known plaintext attack [1] which is something we've dealt with before in several other contests (check out writeup on the Hashes.com contest by Vavaldi, Penguin, and Shooter3k: `https://hashmob.net/writeups/VPS%20-%20Hashes.com%202025%20writeup.pdf`).

```
bkcrack 1.7.1 - 2024-12-21
Archive: @cracktheconcloud/10.0.110.197[CN]_browser_export.zip
Index Encryption Compression CRC32    Uncompressed  Packed size Name
----- ---------- ----------- -------- ------------ ------------ ----------------
    0 ZipCrypto  Deflate     f2cca419         7072         3559 README.txt
    1 ZipCrypto  Deflate     950ff79f          739          174 10.0.110.197[CN]_browser_export.csv
```

With bkcrack (a tool built to exploit the known plaintext attack) it is possible to recover the internal encryption keys by using a known-plaintext attack, given at least 12 bytes of known plaintext. The larger the known plaintext, the faster the attack. We assumed that the README file inside each ZIP archive was (largely) the same as the one we got in the top folder. However, since the README file inside the ZIP archives was compressed, we couldnt simply use the README file we got as plaintext. We first had to zip it (assuming the default compression level was used):

```
zip README.zip README.txt
```

We then used bkcrack with the -D option to create a new ZIP file with the password removed. We need to specify the encrypted ZIP file (-C), the ZIP entry containing the ciphertext (-c), the plaintext file (-p) and the zipped plaintext file (-P).

```
bkcrack -C 10.0.110.197\[CN\]_browser_export.zip -c README.txt
-p README.txt -P README.zip -D nopass.zip
```

[1] Biham, E., Kocher, P.C. (1995). A known plaintext attack on the PKZIP stream cipher. In: Preneel, B. (eds) Fast Software Encryption. FSE 1994. Lecture Notes in Computer Science, vol 1008. Springer, Berlin, Heidelberg. `https://doi.org/10.1007/3-540-60590-8_12`

```
bkcrack 1.7.1 - 2024-12-21
[21:13:44] Z reduction using 3540 bytes of known plaintext
100.0 % (3540 / 3540)
[21:13:44] Attack on 2992 Z values at index 735
Keys: 73a4c9db 48eac8f8 965d3a72
11.4 % (342 / 2992)
Found a solution. Stopping.
You may resume the attack with the option: --continue-attack 342
[21:13:44] Keys
73a4c9db 48eac8f8 965d3a72
[21:13:44] Writing decrypted archive nopass.zip
100.0 % (2 / 2)
```

The whole process is fairly quick and with some scripting we can extract the ZIP file for some of the archives. The CSV file for all archives contains three columns: URL, Email, and Password. Additionally, the Email and Password are all identical with the only difference being the URL containing a clear indication that they did not exist or that the URLs were fake (reducing the chance of a red-herring).

We held suspicion that the email might possibly be re-used somewhere as a username but were unable to find a reference. Based on a post-contest talk the URL and Email only served as flavour to replicate the stealer-logs. However, this approach didnt recover all archives. We were quick to find that the reason for this is that different levels of compression were used, resulting in different 'plaintexts' as compression happens prior to encryption. This means that in order to recover all archives we had to enumerate all possible levels of compression.

The default compression level for the zip command is -6. By creating different zipped versions of README.txt using the compression levels -5, -4, -3, -2 and -1 - and subsequently repeating the process we could recover the remaining archives.

We could then extract passwords from all 5000+ extracted .csv files using a regexp command and ran all passwords against all lists. We received a few hits on the bcryptsalot hashlist and used that to gain over 4500 cracks.

bkcrack also allows you to recover the password that was used to create the encrypted ZIP files:

```
bkcrack -C 10.0.110.197\[CN\]_browser_export.zip
-k 73a4c9db 48eac8f8 965d3a72 -r 12 ?p
```

Like this, bkcrack tries to brute force the password up to 12 characters. However, it quickly became apparent that bkcrack was too slow for this, as longer passwords were used. We eventually figured out that the length of the passwords was 14, a length not usually bruteforce-able but due to the sheer amount of possible permutations. However, using the key obtained from bkcrack we could utilize an optimized attack on the PKZIP master key using mode 20510 in hashcat getting an effective hashrate exceeding an octillion keys per second (depending on hardware).

Our initial attacks were focused on passwords within the ?u?l?d keyspace. Although the first founds did open the ZIP-archives, they did not count as valid solutions due to containing non-printable characters. These non-printable characters were caused by the first 6 characters over which we had no control. By introducing the –keep-guessing flag we could continue to

gather founds without potentially missing the correct solution and we submitted all printable founds to the website.

We later noticed that some of the first 6 characters within our founds contained dots (.) as well which were also accepted by the website. We therefore expanded our attack to also include those within the mask keyspace.

Below are a few examples of our first cracks, which all things considered, is a technical marvel:

```
10.100.233.241[IN]_browser_export.zip:BQqM9ZtkFluVbU
192.168.98.96[CN]_browser_export.zip:bOSiwyhNaANasK
10.100.136.69[IN]_browser_export.zip:tws149pFPprAjb
10.101.204.132[DE]_browser_export.zip:SoCTfCWRgkDWEo
10.102.224.167[GB]_browser_export.zip:oXbtoeWruQNLmo
172.21.232.76[JP]_browser_export.zip:0vWvf3KpoamYSw
```

This list ended up being a solid source of bcryptsalot points and provided a last burst of cracks that helped solidify our first position in the last minutes of the contest.

2.15 WTF_Rob

Hashlist	Plains Found	First Blood	Score
WTF_Rob	105	75	110 700

Although we obtained some lucky hits from this hashlist we were unable to obtain any concrete cracks. As it turns out the hashes themselves were heavily rotated and is an observation we only obtained after attending the post-contest talks. The hashes that were rotated between 0-30x, so you could use } or { [2] to rotate the hashes and create different permutations of it or match it to specific blocks. This modification is something that we are surprised mdxfind ended up missing for us and is a lesson learned. The hashes we obtained were in the subset of 0x rotated which is more luck than skill.

2.16 SaltyHashAlgo3.1

Hashlist	Plains Found	First Blood	Score
SaltyHashAlgo3.1	0	0	0

Unfortunately, we were unable to correctly identify the algorithm for this list before the end of the contest. Post-contest talks revealed that the algorithm that was used for this list was SHA3-256(MD5($PASS)). We wrote a kernel to work on it and have fun after the contest but were unable to recover any during the contest itself. Our lesson learned from this is to also try run hashlists pre-computed with common algorithms to help assist with hash-shucking as an easy way to be cross-compatible with hashcat and other tools.

[2] `https://hashcat.net/wiki/doku.php?id=rule_based_attack`

Chapter 3

Closing Notes

And on that final note we conclude the write-up for this year's rendition of CrackTheCon. We would like to thank the CypherCon organizers, contest organizers (CynoSure Prime), JpG0mez and Goetzman for pushing CsP to make the contest happen again, and finally of course the members of Team HashMob who worked diligently to achieve the results we did. The contest was well-designed, even if it was rushed, and in particular the Zipitydoda and Quorum_Quest were very fun challenges to dive into deeper.

Finally I'd like to invite everyone to check out the `https://hashmob.net/` website and community, and join the Discord community (linked on the website). We are an open community where you can actively research passwords, attacks and learn more about the general field of cryptography. Our community contains members of various backgrounds with a wide variety of skillsets and most relevant questions can be answered expertly.