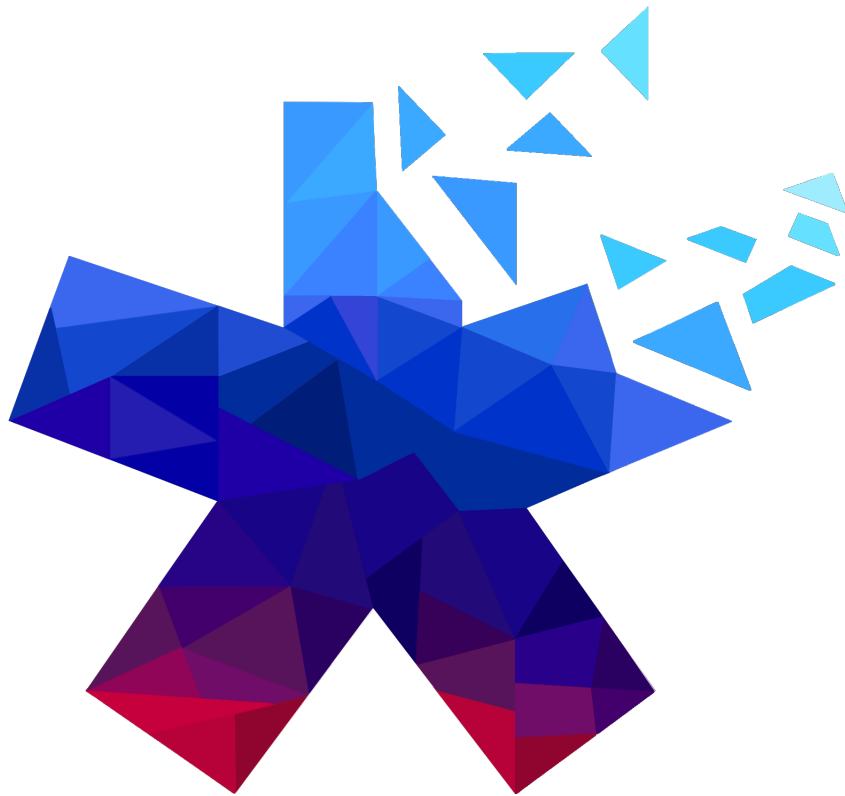


# HashMob

Passwords are like underwear. Don't leave them out where people can see them, change them regularly, and don't loan 'em out.

---

*Twitter, 2007*  
CHRIS PIRILLO



CrackTheCon 2022 Write-up  
HashMob.net  
vavaldi@hashmob.net

# Chapter 1

## Preface

Allow me to preface this write-up by beginning to thank the wonderful members of CynoSurePrime (CSP) for organizing the contest, as well as the amazing members of HashMob.net who contributed to the performance of the teams during the contest itself. HashMob participated with two different teams in this contest. One team for the pro division, and one team for the street division. Although a few of the Pro team had participated in CMIYC 2021 before, a large chunk of them had not played in any contest prior to CrackTheCon, and the Street team had never played together at all. With this in mind the performance and enthusiasm shown during the contest is something to applaud.

The Pro team successfully secured the #1 spot on the leaderboard, marking it the first Win for the team. The Street team landed on the #8 spot but nevertheless performed admirably; gaining many experiences along the way. LevenshteinAmphetamine proved to be an admirable opponent, making it close until the very end. john-users also performed well, recovering more hashes than anyone else in one category. We look forward to participating in more contests in the future, hope to see you all there.

### Pro Team

- Vavaldi
- Cochino
- clem9669
- penguinkeeper
- foordeluxe
- gatete
- Shooter3k
- virodoran
- Just
- \_cin
- w00dsman
- SoSander
- Jimmalina
- Wingman4l7
- Adam Black
- 0x4C00

### Street Team

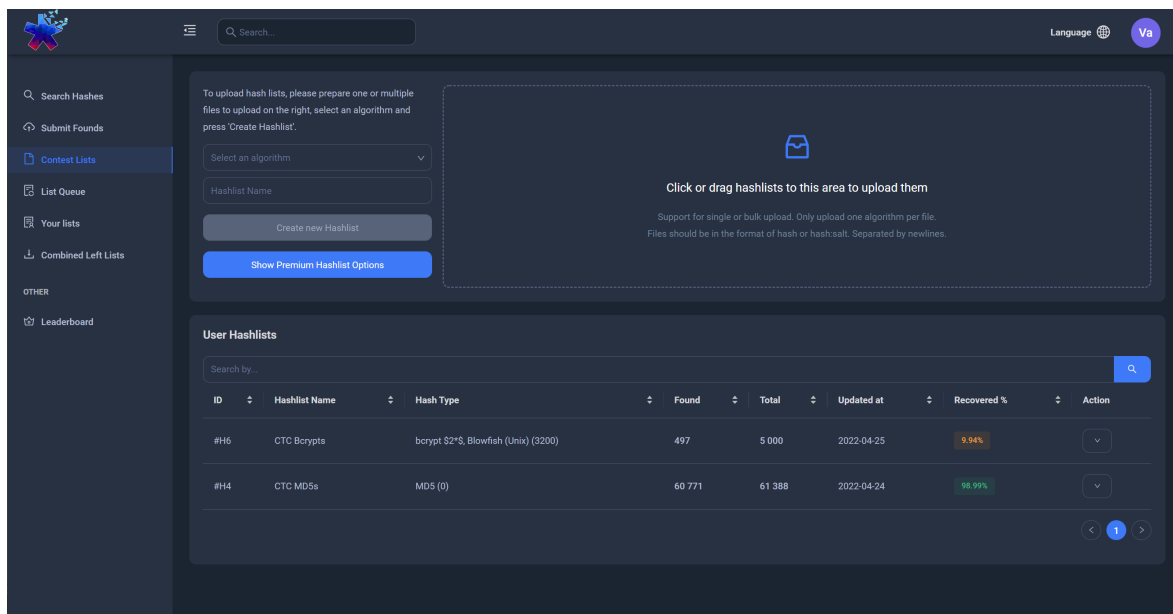
- b8vr
- uwu as a service
- mostwanted002
- m4ulk0rb
- AlwaysAwake

## 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 399 306 307 new passwords / plaintexts and amassed a following of nearly 900 members.

## 1.2 Contest Environments

Each team was given access to a copy of the HashMob.net web-application with a custom back-end script that would use the available API of the contest to automatically submit new found solutions regularly. These environments were restricted in access so only authenticated team members from each respective team could participate. Registration was protected with a secret key which was made aware to each team via their discord channels. i.e. the teams were unable to see each others' hashlists and finds and their applications did not interfere with each other.



The screenshot shows the HashMob.net web application interface. The main content area displays instructions for uploading hashlists, a form to create a new hashlist, and a table of user hashlists.

**User Hashlists Table:**

ID	Hashlist Name	Hash Type	Found	Total	Updated at	Recovered %	Action
#H6	CTC Bcrypts	bcrypt \$2\$, Blowfish (Unix) (3200)	497	5 000	2022-04-25	9.94%	
#H4	CTC MD5s	MD5 (0)	60 771	61 388	2022-04-24	98.99%	

## Chapter 2

# Pro Team write-up

### 2.1 The Preparation

In preparation of the contest the members of the Pro team actively tried to recover as many of the Pro and Street hashes as possible. Partially for fun, partially to see what the CSP team had in store for us. The Pro teams were given one set of MD5 hashes, and one set of bcrypt hashes. The Street teams were given one set of SHA1 hashes, and one set of md5crypt hashes. The rest of the preparation time was spent getting people familiar with a new environment, new scripts and tools were being written which could help us during the contest.

Overview of Demo Lists				
Category	Hash Type	Found	Left	Total
Pro	MD5	60 771	617	61 338
Pro	bcrypt	497	4 503	5 000

### 2.2 Software Used

Listed below are some of the software used by the Pro team, although a majority of them are public / open source tools, some are closed source, modified open source, or specifically developed for the contest. The Hashtopolis instance was modified to automatically submit any founds to the HashMob instance based on the hashtype of the hashlist it was discovered in. We modified the SendProgress API to perform this in a rather 'hacky' fashion and for future instances we would like to improve this (more on that later).

Overview of Used Software			
Name	Open Source	Public	Purpose
Hashcat*	yes	yes	Password recovery
MDXfind	no	yes	Password recovery
HashMob Search*	no	yes	Hash Lookup
Hashtopolis	yes**	yes	Collaboration
HashMob Mirror	no	no	Collaboration
Gramify*	yes	yes***	Analytics
PACK	yes	yes	Analytics
PACK2	yes	yes	Analytics
Plain-Finder*	no	yes	Analytics
RuleProcessorY*	yes	yes	PW generation
PRINCE	yes	yes	PW generation
Audacity	yes	yes	Audio Analysis
sync.py	no	no	Auto submission

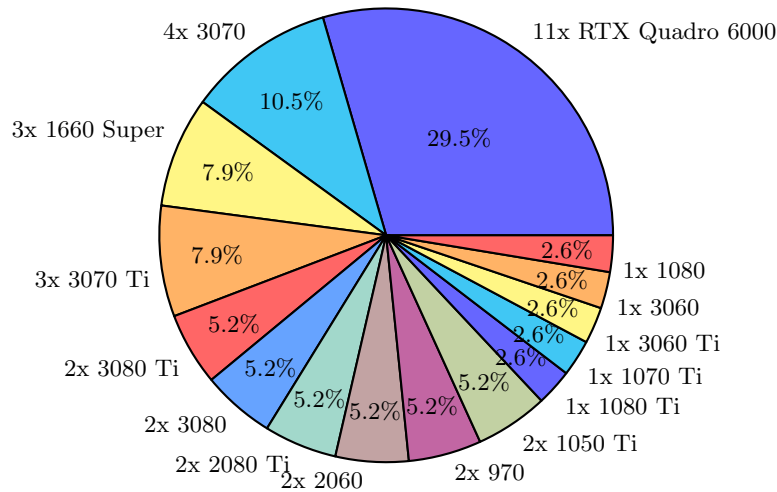
\* These tools can (also) be found on HashMob.net or their discord.

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

\*\*\* The Gramify version used in the contest is in-development and will be released soon

### 2.3 The Hardware

The Pro team has a large show up with a variation of hardware. Some participating with some impressive hardware like the UHD Graphics 620, or the more mediocre 11X RTX Quadro 6000s. A grand total of 31 Graphics cards were used, with a total of 150 CPU Cores of varying brands and Clock Speeds.



### 2.4 Ready? Set...

Go! The first minutes of the contest were hectic, mostly spent identifying which algorithms we were being challenged with. CrackTheCon provided a total of 6 files for both teams to work with. Each list but one was in a .txt format and was easy to start investigating. The other was a .mp3 and seemed to be corrupted or in another form modified with our first

thought being Steganography. With the start of the contest we created a discord channel for each hashlist so that team members could hop between hashlists and keep the chat on-topic and focused on the hashlist. A single General chat was available as well as a 'pins' channel for important announcements and messages.

## 2.5 The Infamous Lists & Results

The lists presented to us for the 2022 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 2022 list yourself, I recommend stopping after this section. The lists we were offered were named: 936 Troubador, Agent Glitch, Gay Melanoma, HC Safe, he-ran-somewhere, numbers^3.

HashMob finished in first place with a total of 3 567 294 points. Closely followed by LevenshteinAmphetamine with 3 208 138 points.

Hashlist	Plains Found	First Blood	Total
936 Troubador	24 524	11 015	29 154
Agent Glitch	42	42	20 040
Gay Melanoma	1426	531	34 999
HC Safe	0	0	13 092
he-ran-somewhere	6 121	2 336	20 682
numbers^3	20 476	18 916	33 000

## 2.6 936 Troubador

Hashlist	Plains Found	First Blood	Total
936 Troubador	24 524	11 015	29 154

This hashlist was interesting. We were initially left scratching our heads when we were presented with a file named 938 Troubador, with a hashlist named 936 Troubador. Until we realized there was a typo, and later identified this hashlist was referring to the popular XKCD comic about password usage (found here: <https://xkcd.com/936/>). Following this, we tried different algorithms such as MyBB and VBulletin with different combinations of the phrase "correct horse battery staple" until we received a hit on a space delimited combination of words with the hashcat mode 2811 for IPB / MyBB (also known as MD5(MD5(SALT)MD5(PASS))). Enumerating further we discovered more words relating to xkcd which resulted in a scraping session of the api XKCD had for their comics in JSON format and building a wordlist on it. In the end, we ended up with 368 base words, some of which were used only partially in the lower count of words (n-grams) and some in the larger count of words (n-grams). The following wordlist gives an overview of the different lengths of plains we discovered:

Words	Plains Found	Unique Words
1 Word	0	0
2 Word	2795	309
3 Word	10337	368
4 Word	11097	368
5 Word	294	99

### 2.6.1 General Approach

Our general approach to this list was first identifying which algorithm was used to hash the plaintexts. After discovering this, we moved on to getting a lay of the land - seeing which base wordlist or source might have been used to recover the hash. For this list, we identified that the xkcd website and comics were a core source of information, and moved to try both xkcd scraped lists as well as common English words, slowly working up our list and having Hashtopolis do the heavily lifting of combining all plaintexts. To create our wordlists we primarily made use of the combinator.exe and combinator3.exe tools which are part of the hashcat-utils toolkit.

### 2.6.2 Timeline

This section gives an overview of the timeline, listing (major) events with time notation as HH:MM.

- 00:00 Contest Starts
- 01:56 First Blood by Vavaldi
- 03:29 Discovery by \_cin that 2 and 3 word combinations were possible (not just 4+)
- 48:00 Contest Ends

### 2.6.3 Learning Points

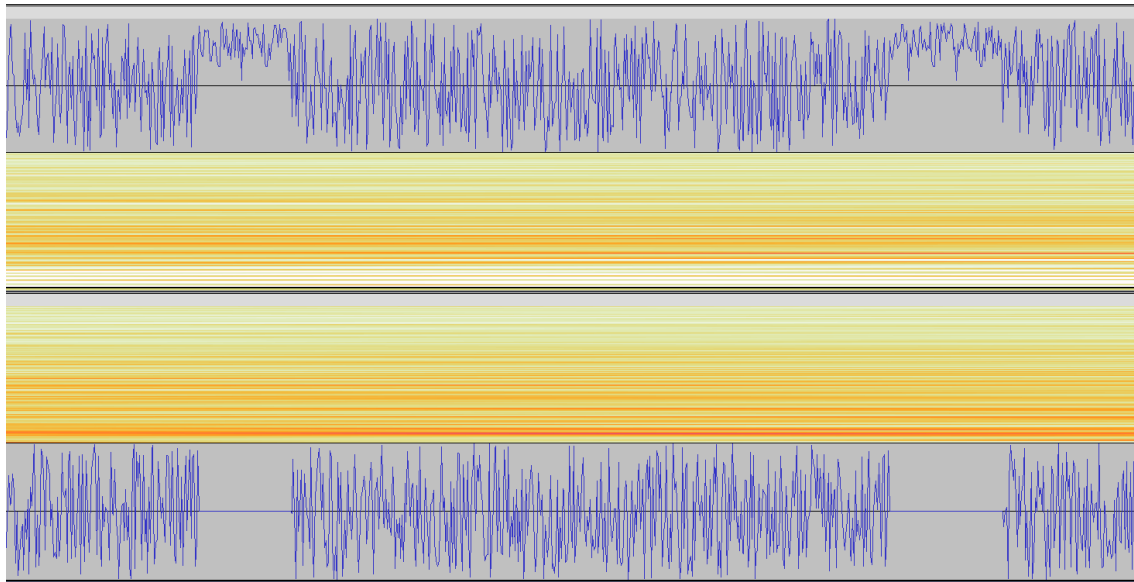
This list was fairly straightforward. It took a little while to identify the algorithm since we were busy with other lists and didn't get hits with simple wordlists until we discovered and properly worked out the XKCD reference. The later discovery that 2 and 3 word combinations were possible was a surprise moment to quite a few working on the list as no one had bothered trying them yet - working through them quickly expanded our base wordlist.



## 2.7 Agent Glitch

Hashlist	Plains Found	First Blood	Total
Agent Glitch	42	42	20 040

This hashlist was presented to us as an .mp3 file, which already confused a few of us. Viewing the file in a text editor or using the 'strings' linux command showed that portions of the file contained Wordpress / PHPBB3 hashes strewn throughout the file, grouped together in chunks. Extracting them with RegEx allowed us to recover a total of 5 hashes which gave the plains: "mix" and "salt". This gave us the thought that the salts of the hashes might be mixed around. Searching the name Agent Glitch brought us to a SoundCloud from hOffman, who authored the song "Agent Glitch" (A hint later verified this). While importing the two files in Audacity as raw data and displaying their audio, a note was made of the fact that both audio files lined up perfectly. The extractable hashes overlapped perfectly with the null bytes shown between the frames of MP3 files and the following image illustrates this with the top image being the encrypted file, and the bottom file being the original audio file from SoundCloud.



Following this realization, we were able to quickly decrypt the file by XOR'ing the given file against the original and obtained a valid file with 20040 hashes. We then started recovering more plains which gave more hints to what the rest of the hashes might hold, in total we recovered an extra 36 hashes here for a total of 40 - leaving 20k. The idea that they might be mixed salts, modified, reversed or otherwise transposed was suggested and a lot of combinations were tried. Due to the low cost function we were able to process a higher amount of plains than otherwise possible, but with 20,000 \* 20,000 possibilities leading up to a 13.4GB file of PHPASS hashes, we were unable to make much progress and kept coming back to the list to try and figure out what changes were made on account of the fact that this list held the most plaintexts and we had already cleared the first hurdle - something none of the other teams had done yet, this gave a great advantage if we were able to unlock the other plains and could even win us the contest with this list alone. Due to the amount of possibilities and little actual hits we were receiving we did not commit to our original idea in full and instead

focusing on trying a lot of different ideas with half effort. Once the last hint dropped from CynoSurePrime we were determined to discover what the exact plains were and after trying a significant portion of the combination of salts x hashes we discovered our first 'real' plain, and a few hours later our second. With the discovery of our second hash we attempted to find correlations between the original line numbers of the hash and salt to see if there was a correlation between them, but we were unable to find one at the time, and though not clearly answered we do not believe there is one. All-in-all the list was a difficult one to complete in the given time-frame and might have gone better if we had either committed to our initial thoughts, or were faster with discovering the XOR method to retrieve the original file.

### 2.7.1 General Approach

Our general approach to this list was spent analyzing and trying to come up with ideas. For the file we tried some extractions and modifications to see if some type of steganography was used to hide the file within another - but nothing ended up working on that front. When the original song was discovered, the suggestion was made to compare the file to the original at a bit level, but it was left at that and not further investigated (missed opportunity). Once the songs' relevance was officially confirmed, interest came back to the list and strides were made in recovering the original file, yet no progress was made after for the simple reason: nothing returned a 'quick' result. Trying out a lot of different things from cutting up the hash to rotating it, modifying the way encode64 vs base64 worked within the hash function to see if the 'mixing' and 'shaking' of the encode64 function in PHPASS had anything to do with the hints from the hashes. Trying different permutations of H / P and rounds from up to 10 but without a single hit throughout our attempts we were ultimately just stabbing in the dark.

### 2.7.2 Timeline

This section gives an overview of the timeline, listing (major) events with time notation as HH:MM.

- 00:00 Contest Starts
- 00:16 Initial 1650 hashes extracted by penguinkeeper
- 00:30 First Blood by w00dsman
- 01:31 Discovery of the original soundcloud song by penguinkeeper
- 19:16 CSP Released hint with song
- 19:22 Song mp3 was first shared
- 20:03 Discovery of bit similarity in songs by Vavaldi
- 20:12 20040 hashes extracted via XOR by Vavaldi
- 20:13 'Second' Blood by penguinkeeper with new plains
- 41:47 Second hint with "mix/shake salt order" and 1 correct hash+salt combination
- 43:19 First real crack 'suilek' by Jimmalina

- 47:06 Second real crack 'iauuku' by Jimmalina
- 48:00 Contest Ends

### **2.7.3 Learning Points**

The overall difficulty of the list was a bit too steep at 400,000,000 possible initial hashes with relatively difficult plains. Once more plains are discovered, the list would get increasingly easier as each found would eliminate 19,999 other hashes. Sharing the MP3 file with the team sooner could've led to an earlier discovery of the XOR and subsequently given us more room to take our time enumerating the salts.

### **2.7.4 Final note**

A final note about this hashlist is that although it was difficult, it was also ingenious and fun with the use of XOR to hide the original file but still have some hashes available to crack (be it intentional or not), and the shifting of salts around. With a slight modification to the plain distribution, algorithm or salt count (using duplicate salts) this hashlist could have been very reasonable.

## 2.8 Gay Melanoma

Hashlist	Plains Found	First Blood	Total
Gay Melanoma	1426	531	34 999

What a name, that surely can't be right. Some of the first thoughts given by our members on the list, suggesting it might be an anagram (which it was: "Anomaly Game"). We never truly discovered what was wrong with this list and simply went through it with sheer luck by using wordlists and rules for the most part. The real solution - based on the after-party discussion with a staff member of the contest - was that the hashes provided to us were off by one or multiple bits/bytes. Hashcat can crack a portion of these hashes due to the method in which it checks hashes and doesn't require them to be perfect matches - but this has a limit. During the contest we mostly threw it on the pile that the hashlist was hard and we were yet to discover a pattern.

### 2.8.1 Learning Points

Our primary learning point for this is that we could have analyzed and compared the input and output of correctly retrieved plaintexts to verify if they were completely correct.

## 2.9 HC Safe

Hashlist	Plains Found	First Blood	Total
HC Safe	0	0	13 092

Although a lot of attempts were made, nothing stuck. In the end, even the hint given for the hashlist turned out to be incorrect with the algorithm being listed as sha256. With the solution being: keccak-512 with plains which were  $\geq 256$  characters. These were picked from places like 'Hashes.org Junk' or heavily used the 'd' rule. When approaching this hashlist we tried many different forms and iterations of SHA256.\* and SHA512.\* with -b -e -i -g and more within MDXFind, with the idea that the plaintexts might be unicode encoded, rotated or truncated to the point where hashcat would not properly recognize them. The (128 character) hashes were split up into two sections of 64 characters and each ran individually as different forms of SHA256. Eventually we spent little time on it with the knowledge that other teams did not get any plains either, and that the list itself was worth relatively few points.

### 2.9.1 Learning Points

Our primary learning point for this is that we could have ran our most common wordlists on all algorithms matching the length of the hash which would ultimately include keccak-512.

## 2.10 he-ran-somewhere

Hashlist	Plains Found	First Blood	Total
he-ran-somewhere	6 121	2 336	20 682

The algorithm was initially presumed to be PBKDF2-sha256 (Django, 10000) due to our familiarity with the algorithm but was corrected to match Python's passlib pbkdf2-sha256 (20300). The plains we initially discovered pertained to what seemed to be gaming focused things involving minecraft, political persons and ransomware names. Discovering the reference to the joke and understandings its concept (being: he ransomware) we started to scrape different sources for ransomware names and running them against the list, slowly getting more hits but still missing a significant portion of founds. Our first breakthrough in discovering how to attack this list was the realization that the hashlist was sorted by the alphabetical value of the plaintext. This meant that we could focus our efforts on specific prefixes or subset of plains. However with very little plains discovered, we were unable to fully start utilizing the plains. This picked up more later when we started grabbing the already cracked plains and 3-5 hashes above and below that plain and running an association attack with rules. The below image shows an example of how our early file looked with plains discovered. This view of the file was generated by a special API endpoint on our collaboration server which we used in a later stage to help automatically create candidates but essentially takes the original file and appends :<plain> to the matching hash.

### Small insight on association attacks

Association attacks are precision based attacks instead of brute force. Instead of checking a thousand, million or billion combinations against a single hash you check a single candidate which you know or assume to be relevant to the targeted hash. A salted hashlist requires the program to re-hash every plaintext you wish to try for each unique salt (this is what makes salted hashes difficult). By using an association attack you are essentially limiting the attempts to 1 plaintext per 1 hash. The keyspace of the attacks are as follow where  $p$  is amount of plaintexts in the wordlist,  $c$  is the amount of hashes and  $s$  is the amount of salts:

- Dictionary attack on MD5:  $p * c$
- Dictionary attack on Salted MD5:  $p * s * c$
- Association attack on Salted MD5: 1

With association attacks (-a9 in hashcat) only one plaintext or 'candidate' is tried for every hash. This is then multiplied by the amount of rules used. Currently, only the latest beta of Hashcat supports rules on association attacks, therefore it will be part of hashcat 6.2.6 or 6.3.

```

8519 $pbkdf2-sha256$1000$6r33/r937j2HUGqNkFJeSw$12/40Tnp6MkKvWtXuw6hagAhb4V1NCh/65re6aateWs:↓
8520 $pbkdf2-sha256$1000$kfKek3JOKSVkjDEGwLjX.g$YewPhT9NVNItYVtb..fQKsDEGL2JGxzY7BxCiDZ1M7A:↓
8521 $pbkdf2-sha256$1000$VfKew8NIYQ2vv/v5dyrg$VPRFgTJ.Fxu0jM5mQ6jKXKDXjttN518F81iB2MFCwX8:↓
8522 $pbkdf2-sha256$1000$xljrvTFGGMN4b835HZMs5Q$YvAe80B0CCPtGJG0m88c8tceBZFWVs09VwFobCsEK3E:encryptoJJS↓
8523 $pbkdf2-sha256$1000$svZ.D6G01rJW6j1HyBmjVA$XoqETTm.46UdG0yx9xECi8N.O.weMPsDnwZk0FSLP7Q:↓
8524 $pbkdf2-sha256$1000$ovS.1zrn/P9/792bM6a0dg$IIiixRKXDeJDIRhnsYuUtF/p9tLC3.wZi2s2deuGCA:↓
8525 $pbkdf2-sha256$1000$07r3nnNu7Z3znpNjyGydg$7nVL/kC5ngBhnqpbEBJyqF.21mNg2X3LLZixSp4wZv4:EncryptoJJS1980↓
8526 $pbkdf2-sha256$1000$u1eqtXa0EeK8FwKgnIYQog$7JhLTTaQ8jY8DODZD8EQMaEBSiW7kLkD/4U3fr5iAL4:↓
8527 $pbkdf2-sha256$1000$977X2nsPobSWSlaKMUaIEQ$ru0kku0QMOfLIE2ovhL69wHwTqVC8NsArOfFVB2P1iI:EncryptoJJS980↓
8528 $pbkdf2-sha256$1000$qqUy7q1U0ue8F0LI.b.Xkg$LdhVekMA6p0Q/InjNVZJPiZIKQoxRdNC3LYr.LI0DZo:↓
8529 $pbkdf2-sha256$1000$2Nub01rL0wcm4Xxvba2VEg$MwqVUUjPqBGx90imUVfmOLiIEY9/HT1PsLb6srhPCv0:↓
8530 $pbkdf2-sha256$1000$.Z9zr1XKKea.111LqHa001g$JGeYoxey7E85htJpdARIKZXfGwPtUKBAYIXQ.0poNys:↓
8531 $pbkdf2-sha256$1000$xxJpMVPKmb0MwMzMScmZkw$SAjwaopzk3oprqwDZkQZQD1escKdyUqMXLCGROT3KSA:↓
8532 $pbkdf2-sha256$1000$F2KstRYCwLh3D5GYMwYghA$IMZkImyZF4qej4k4m1aAULtX11t5XFkQFjItPcyHFv0:↓
8533 $pbkdf2-sha256$1000$CKGUSTaac26t9X5v7f1fCw$g59v5qYzOLGI.RhpMC1CxT5PrZIw7ytX/yIHdxTjama:↓
8534 $pbkdf2-sha256$1000$RohRG6h1jzDkHYGwt5RkDkA$HDZ0eVYD7DneKzsw2rSuuEbrYmdeSS2vWJUq79N4Ww:↓
8535 $pbkdf2-sha256$1000$VIpx7r0XwphTyh1DqFXKGQ$5V70eAS47Angrej0D7C8FUGH.a3rh1YnQ0W7U9aDum:↓
8536 $pbkdf2-sha256$1000$C6F0bu1dq5VyzhnjvFdqTQ$aN9aBi8shbW35NpvpF0Iq1HFwUo3bNyr0Imtb33U4o:↓
8537 $pbkdf2-sha256$1000$LmUxsh7L.UcIwQgRIjxXg$Fjb8Zc8tg4itQ1bxQMBn7sTVbwVA8WThc15wFipPFs:↓
8538 $pbkdf2-sha256$1000$/Z9TSul9r3UuhdD631urdQ$H43XUrSycLDZQnDXbKAiUFe8580NypuC.h58CJtCALQ:↓
8539 $pbkdf2-sha256$1000$08e4FwKgvCoFIOR87907Zw$1XAC6RgVI618BIvorduMZg9/3N1gCepoLLpN29NwSfu:↓
8540 $pbkdf2-sha256$1000$0VprLQXAmJ0y11JqrdUa4w$0/lyo4CRND4CNmoHwAvvaBmej1P1pZwyyQsI..L4:Enigma↓
8541 $pbkdf2-sha256$1000$h9B6771XSqnVgPAmJPSug$w1WISjeQSKYwteGmtS2J/fDx/qIGReRI1X75u771Gum:Enigma541↓
8542 $pbkdf2-sha256$1000$PieEEI0RajvGiPk/F/rfQ$eeRxxbeuV48Pghs36Aj5UBY2Bb3vLcRkMU41/0VT1Z4:↓
8543 $pbkdf2-sha256$1000$RqHBCOH8/58zJgQAYEYJM$SoD.FIoUmKgy7p3c8dQ4zY4pTjObX11h1n5hfw4yoHI:↓
8544 $pbkdf2-sha256$1000$dE5JCaE0RgihdC71XCullA$yfnWRk33gn8G50GiPYej6GvAVuX2NiuV8Wd1kLH9i4o:Enim3a↓
8545 $pbkdf2-sha256$1000$1jLmPGFMGWmSJaT03jnsnZA$ghX180KdDC5ArWUYZwQW8Wxo/STcBHKS..XottV4nQw:↓
8546 $pbkdf2-sha256$1000$gDAmxFhrzXmPEWJMyTknxA$4FQGI6..co3VP/ZdHOVBzhMHUN8M/TQmmXmQuRjJZC8:↓
8547 $pbkdf2-sha256$1000$LaWUMqYUQmhnA2p9Z4T4g$3P0uPfyux/B/K6tPOEotk2usNjUXfM0sJmENV2K9yic:↓
8548 $pbkdf2-sha256$1000$UMoZQ6i1fKJIYRmDcC51bA$dinD/3AlrcYcZ5gJ88kkR5qVU0uxNaIuH7loABRK8n4:↓
8549 $pbkdf2-sha256$1000$g/A.ByBEiDFGKIw1hqtA$82W8Y93sKtYpvxrxRsQKFR48vwJV7NUhHVqpPVI0Ius:↓
8550 $pbkdf2-sha256$1000$gzCm9L4XYmzNuTcm5DxnrA$S520nEHS0g1DUUBmrmBFZQYmhRcRpfI6qemH/ixY7.Y:↓
8551 $pbkdf2-sha256$1000$USplDCHkXESJYax1jtGaMw$uyMYmUVEkG10tE7vbL4qTRH4jSiXzRB25MHCKKw/3Lo:↓
8552 $pbkdf2-sha256$1000$.f/fw/BuFcKYcw559w7Bw$KJepip7Maj6666drQnWT2mJibVwHk402TO.8uMMa54:↓
8553 $pbkdf2-sha256$1000$zZlHiLG2VkopRUipFQKAcA$jMKdguBnPMtZuVqI7Ltp10NsQDf95SyJ2/IGux677f0:↓
8554 $pbkdf2-sha256$1000$Y0wdA0DIufe.lxJiDCEE4A$yM88qzLNQ3ee4nw3UaMhw/6/yCZZPetlkkNte36SzqE:Enjey↓
8555 $pbkdf2-sha256$1000$uNc6B4Dw3rtXqrU2phSiFA$S8XVxYuXHHW64JViy3x8pwul6nQ25Dxj0JP9CbXPaIg:↓
8556 $pbkdf2-sha256$1000$61nr7VulLXndD6v3fuPQ$RnHhW6E2TC3c0QA3eRlwmR8uW0n5kneut7174nhr1T:↓

```

Our aim and method was straightforward:

1. Find an already recovered plaintext hash in the original list.
2. Take the N hashes above and N hashes below the 'found' hash in the original file.
3. Create an association attack hash and candidate for those hashes with the candidate being the plaintext of the previously found hash
4. Run the association attack with rules to find plains similar to the one found initially
5. Repeat

Large strides were made with this hashlist when a script was released which would automate this process up to the last step and basically give every team member the ability to run their own rules and modifications on the limited candidates, instead of on everything. This version of the script was later improved upon to reduce the amount of duplicate founds and further optimize workflow of members. The script below demonstrates how we generated our files.

```
import requests
import json

api_key = "API_KEY_OF_USER"

def fetch_analytics():
    headers = {"api-key": api_key}
    r = requests.get(
        'https://$domain$/api/v1/hashlist/5/analytics',
        headers=headers
    )
    return r.json()

left_hashes = []
sorted_data = {}
candidate_set = []
data = fetch_analytics()['data']

f = open("ransom_founds_overview.txt", "w+")
for key in data:
    sorted_data[int(key)] = data[key]
    if data[key]['plain'] is None:
        f.write(data[key]['hash'] + "\n")
        left_hashes.append(data[key]['hash'])
    else:
        f.write(data[key]['hash'] + ":" + data[key]['plain'] + "\n")
f.close()

for i in range(1, len(sorted_data)+1):
    if sorted_data[i]['plain'] is not None:
        for j in range(-10,10,1):
            if i+j < 1: continue
            if i+j > len(sorted_data): continue
            if j == 0: continue

            if(sorted_data[i+j]['hash'] in left_hashes):
                candidate_set.append({
                    "hash": sorted_data[i+j]['hash'],
                    "plain": sorted_data[i]['plain']
                })

f = open("ransom_founds_hashes.txt", "w+")
g = open("ransom_founds_candidates.txt", "w+")
h = open("ransom_founds_combo.txt", "w+")
for _set in candidate_set:
    f.write(_set['hash'] + "\n")
    g.write(_set['plain'] + "\n")
    h.write(_set['hash'] + ":" + _set['plain'] + "\n")
f.close()
g.close()
h.close()
```

The following line indicates how we specified to grab the 10 lines before and 10 lines after the intended hashlist. Initially we kept this at -50 and +50 to ensure that we didn't immediately miss out on hashes. The longer our attacks continued, the more we slimmed this down to 15, 10 and 5.

```
for j in range(-10, 10, 1):
```

In the end our list ended up looking similar to the following image, where it is clearly noticeable how the list was generated and given more time I am certain we could have recovered a larger portion of hashes.



### 2.10.1 General Approach

Our initial approach to this hashlist was to discover as many plaintexts as possible though the use of small wordlists like Hashmob Combined micro and Hashmob Combined mini (from Hashmob.net), getting a large chunk of the base words. When we discovered the relevance in alphabetical order we saw the potential for more optimized attacks and were later able to utilize this. Seeing how many points this list was worth the last few hours was spent by all members running different types of attacks and rules. Digit appends were found to be one of the most effective ones and rule-stacking with debug-mode/debug-file enabled was utilized heavily to help identify patterns. These files were later exchanged, optimized and sorted to then be re-ran with stacked rules like nsa64 and best64. Additionally the Gramify tool was used with character-based n-grams to create popular suffixes within the file.



### 2.10.2 Timeline

This section gives an overview of the timeline, listing (major) events with time notation as HH:MM.

- 00:00 Contest Starts
- 00:03 First Blood
- 00:17 Reference found to the joke made at #PentestHackfestEU by clem9669
- 26:59 Discovery of (mostly) alphabetically ordered hashlist by plaintext by Vavaldi
- 40:41 First -a9 attack building script with -20, 20 range released by Vavaldi
- 48:00 Contest Ends

### 2.10.3 Learning Points

Our main learning point for this hashlist was that we could have had a slightly better focus on the hashlist and be faster to note it was not sorted the same way as other lists (which was alphabetically by hash). Unfortunately hashtopolis does not support association attacks (yet), so making use our combined hardware was limited. We attempted to run some wordlist attacks with rules based on the founds (small amount), which yielded in a few cracks but limited amounts.

## 2.11 Numbers<sup>3</sup>

Hashlist	Plains Found	First Blood	Total
numbers <sup>3</sup>	20 476	18 916	33 000

numbersnumbersnumbers (Numbers<sup>3</sup>) was an MD5-based list with many mixes of iterations. The plaintexts were built up of three digits in front of passwords followed by the passwords, often (but not always) appended with numbers. After running MD5 for a short period with rules we discovered the similarity in prefix and ended up running `?d?d0` prefix rules to discover more plaintexts. As time went on we discovered a correlation between the iteration counts and prefixes and were able to further hone in on the larger MD5 iteration counts. We primarily utilized MDXFind for this with the `-i` flag.

### 2.11.1 General Approach

Our general approach to this list was making use of the rules and derived prefix rules to target the hashlists. With the finding of iteration counts we analyzed the plains and found a correlation between the iteration count and prefixes which we could then use to further target the larger iteration counts more effectively.

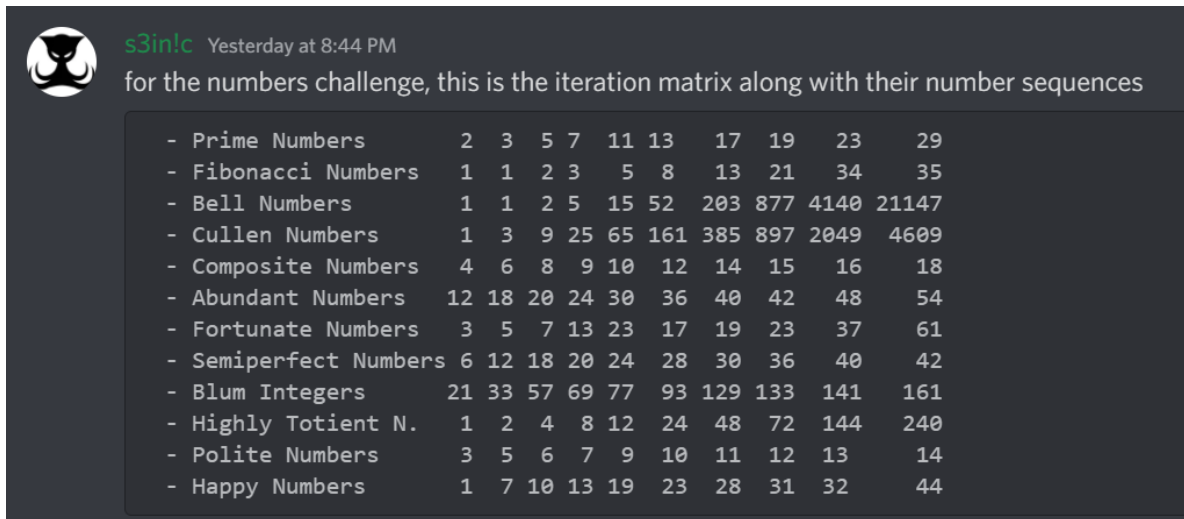
### 2.11.2 Timeline

This section gives an overview of the timeline, listing (major) events with time notation as HH:MM.

- 00:00 Contest Starts
- 00:17 First pattern discovered with `?d?d0` prefix by `_cin`
- 08:16 First indication of MD5x2 or more being used by Shooter3k
- 08:22 First indication of MD5x3 or more being used by Shooter3k
- 08:25 First indication of MD5x4 being used by w00dsman
- 08:28 First indication of MD5x7 being used by w00dsman
- 08:31 First indication of MD5x9 and less being used by w00dsman
- 17:13 First indication of MD5x141 and less being used by Vavaldi
- 17:15 First indication of MD5x240 and less being used by Vavaldi
- 17:37 First indication of MD5x897 and less being used by `_cin` and Vavaldi
- 17:45 First indication of MD5x4140 and less being used by Shooter3k
- 18:44 First indication of MD5x21147 and less being used by Shooter3k
- 18:48 Realization that each iteration count of MD5 had specific prefixes by Vavaldi
- 41:12 Custom MD5x4 to md5x199 Pure kernels for hashcat released by `_cin`
- 48:00 Contest Ends

### 2.11.3 Learning Points

The list ended up being built up with mathematical functions and popular number sequences, something we did not discover until after the fact. The image below shows the sequences used by the contest. When working on the list we could have better analyzed the iteration counts of the hashlist to identify which specific iteration counts were being used and which corresponding prefixes belonged to them.



s3inlc Yesterday at 8:44 PM  
for the numbers challenge, this is the iteration matrix along with their number sequences

- Prime Numbers	2	3	5	7	11	13	17	19	23	29
- Fibonacci Numbers	1	1	2	3	5	8	13	21	34	35
- Bell Numbers	1	1	2	5	15	52	203	877	4140	21147
- Cullen Numbers	1	3	9	25	65	161	385	897	2049	4609
- Composite Numbers	4	6	8	9	10	12	14	15	16	18
- Abundant Numbers	12	18	20	24	30	36	40	42	48	54
- Fortunate Numbers	3	5	7	13	23	17	19	23	37	61
- Semiperfect Numbers	6	12	18	20	24	28	30	36	40	42
- Blum Integers	21	33	57	69	77	93	129	133	141	161
- Highly Totient N.	1	2	4	8	12	24	48	72	144	240
- Polite Numbers	3	5	6	7	9	10	11	12	13	14
- Happy Numbers	1	7	10	13	19	23	28	31	32	44

## 2.12 Summary

All in all, it was a very good experience for everyone to participate in this contest and many are already looking forward to participating in the next; competing against some of the same teams hopefully. Our communication and analytical skills are definitely a factor which can use some improvement. Although the majority of users have quite a bit of experience in hash cracking, many aren't used to the non-standard format of contests and still have to get familiar with some of the concepts presented to them.

## Chapter 3

# A Free Lesson

Thank you for reading through the write-up and getting to this point. Before we continue on with the Street Team, I wanted to share with you an attack method that I just came to realize, very few people are aware of, that can help you quickly harvest founds with little effort. It's called `-loopback`, some of you might already be familiar with this flag, yet I suggest you keep reading. Hashcat can make use of rules, simple additions, subtractions, transpositions or other modifications to a word in a wordlist attack, when using these we refer to it as a 'rule attack'. An example of this is the `l` rule which makes all letters lowercase in a word, or the `il1a` which would "i", insert a letter "a" after the "l"st character of the word. Modifying the word "password" to "paassword". When you've ran some rule attacks and discovered some plaintexts you can introduce `-loopback` to your rule attack to not only run your rule attack on your wordlist, but also on all your previously found plaintexts, which means you don't have to extract them and then run rules on them. This saves time, and if your loopback finds any new hits, it'll start another loopback with those new founds and so on and so forth until it's ran all your rules on the last few plains with no new hits and then stops. With the right rules this can be an incredibly powerful attack and I highly recommend trying it with some of our community's rules (check [hashmob.net](http://hashmob.net) for accurate suggestions). We recommend rules like `fordy10k`, `fordy50k`, `fordyv2`, `Robot_MyFavorite` and `top_5000` to `top_250`.

Now for the thing most people aren't aware of: When you just want to run some different rules on your found plains you might extract them and then run rules on them but instead, what you can do is create a new `.txt` file and name it `empty.txt`. Put nothing in it, and specify it as wordlist when doing a rule attack with `loopback`. Hashcat will skip over the empty wordlist and instantly start your loopback attack. This means you can run a lot of different rules over your found plains to try and abuse similarities between already found plaintexts without any drawback or increased keypace you'd get from running `-loopback` with an extract wordlist, or with running another (potentially small) wordlist.

## Chapter 4

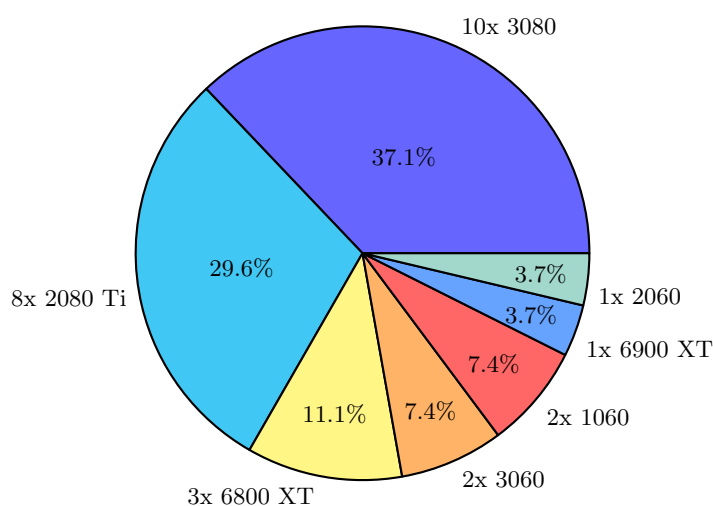
# Street Team write-up

### 4.1 The Preparation

In preparation of the contest the members of the Street team tried to recover as many of the Street hashes as possible. Partially for fun, partially to see what the CSP team had in store for them. The Street teams were given one set of SHA1 hashes, and one set of md5crypt hashes. Some of them were entirely new to HashMob or hash cracking itself so some orientation was required. The tools and scripts developed by the Pro teams were actively shared with them and for the duration of the preparation multiple pointers and tips were given to help guide them to perform more optimized attacks.

### 4.2 The Hardware

The Street team had an extremely powerful show of hardware, exceeding the overall power of the pro team with a grand total of 27 graphic cards, with a total of 66 CPU cores.



### 4.3 The Infamous Lists & Results

The lists presented to us for the 2022 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 2022 lists yourself, I recommend stopping after this section. The lists we were offered were named: 637970686572636F6E, Challenge 1 up to Challenge 9, Archives. The Archives challenges refer to the two encrypted archives from which hashed passwords could be extracted. Only one of these was successfully opened which contained the Challenge 7 hashes.

HashMob finished in eighth place with a total of 829 236 points. With the first place by 'toil' having 2 159 197 points.

Hashlist	Plains Found	First Blood	Total
637970686572636F6E	46 973	0	79 975
Challenge 1	11 721	2 037	24 063
Challenge 2	0	0	11 334
Challenge 3	32 871	412	95 004
Challenge 4	7 661	16	25 180
Challenge 5	0	0	6 454
Challenge 6	159	61	43 609
Challenge 7	122 999	12 554	239 127
Challenge 8	19 277	25	?
Challenge 9	188	74	12 180
Archives	2	0	2

## 4.4 637970686572636F6E

Hashlist	Plains Found	First Blood	Total
637970686572636F6E	46 973	0	79 975

### 4.4.1 Identification

The general approach was to use `hashcat --identify 637970686572636F6E.txt` to list out all possible hash types. After a list of hash modes was obtained, we moved to hit-n-trial approach to identify the hash type by using hashcat dictionary attack mode with `rockyou.txt` as a base dictionary. As soon as we got more than 2 plains recovered, we confirmed the hash type for the challenge. For this particular challenge the use of SHA1 was confirmed.

### 4.4.2 Attack Approach

At very first, we tried basic dictionary attack with popular wordlists `rockyou.txt` and `crackstation.txt` to recover as much as possible before moving onto advanced attack methods. Once we had sufficient plains recovered and dictionaries were exhausted, we analyzed the pattern found in them. The majority of words were comprised of **lowercase letters and numerical characters**. We generated mask in combinations of `?d` and `?l`. Apart from mask based attacks, we restarted dictionary based attacks with new found plains included and adding in popular rules.

- Attack modes used: Dictionary (with rules) + loopback, Masked brute-force
- Wordlists used: `rockyou.txt`, `crackstation.txt`
- Rules used: `pantagrul.one.royce.rule`, `leetspeak.rule`, `nsa64.rule`
- Mask generator used: `maskgen`

### 4.4.3 Learning Points

Initially we considered the hashes to be plain MD5 and started attacking, which led to wastage of some of our time. When the actual hash type was identified, things were resolved faster, hence confirming identification before throwing resources and time at a challenge is wiser.



## 4.5 Challenge 1

Hashlist	Plains Found	First Blood	Total
Challenge 1	11 721	2 037	24 063

### 4.5.1 Identification

The general approach was to use `hashcat --identify Challenge1.txt` to list out all possible hash types. After a list of hash modes was obtained, we moved to hit-n-trial approach to identify the hash type by using hashcat dictionary attack mode with `rockyou.txt` as a base dictionary. As soon as we got more than 2 plains recovered, we confirmed the hash type for the challenge. For this particular challenge the use of MD5x2 was confirmed.

### 4.5.2 Attack Approach

At very first, we tried basic dictionary attack with popular wordlists `rockyou.txt` and `crackstation.txt` to recover as much as possible before moving onto advanced attack methods. Once we had sufficient plains recovered and dictionaries were exhausted, we analyzed the pattern found in them. The majority of words were comprised of **lowercase letters and numerical characters**. We generated mask in combinations of `?d` and `?l`. Apart from mask based attacks, we restarted dictionary based attacks with new found plains included and adding in popular rules.

- Attack modes used: Dictionary (with rules) + loopback, Masked brute-force
- Wordlists used: `rockyou.txt`, `crackstation.txt`
- Rules used: `pantagrule.one.royce.rule`, `leetspeak.rule`, `nsa64.rule`
- Mask generator used: `maskgen`

### 4.5.3 Learning Points

Initially we considered the hashes to be plain MD5 and started attacking, which led to wastage of some of our time. When the actual hash type was identified, things were resolved faster, hence confirming identification before throwing resources and time at a challenge is wiser.

## 4.6 Challenge 2

Hashlist	Plains Found	First Blood	Total
Challenge 2	0	0	11 334

### 4.6.1 Identification

We were not able to identify the hash type for this challenge. We used the approach similar to Challenge1 for identification and hence failed. The hash type was revealed after the competition, turned out to be `md5(md5(md5(md5(md5(pass))))))`, which is not available in Hashcat.

## 4.6.2 Learning Points

Don't limit testing to a single set of software. If one tool exhausts, try similar tools and tinker with their available options.

## 4.7 Challenge 3

Hashlist	Plains Found	First Blood	Total
Challenge 3	32 871	412	95 004

### 4.7.1 Identification

The general approach was to use `hashcat --identify Challenge3.txt` to list out all possible hash types. After a list of hash modes was obtained, we moved to hit-n-trial approach to identify the hash type by using hashcat dictionary attack mode with `rockyou.txt` as a base dictionary. As soon as we got more than 2 plains recovered, we confirmed the hash type for the challenge. For this particular challenge the use of `SHA3-512` was confirmed.

### 4.7.2 Attack Approach

We kept the same attack approach, i.e. recover partially using basic wordlists and rules, analyze and proceed to advanced attacks. The pattern was identified as names of places and people.

- Attack modes used: Dictionary (with rules) + loopback, Masked brute-force
- Wordlists used: `rockyou.txt`, `crackstation.txt`
- Rules used: `pantagrule.one.royce.rule`, `T0X1cv1.rule`, `leetspeak.rule`, `nsa64.rule`
- Mask generator used: `maskgen`

### 4.7.3 Learning Points

Increasing the scope of wordlists, and improving pattern recognition helps in narrowing down the target wordlist and recovers larger number of plains in less time.

## 4.8 Challenge 4

Hashlist	Plains Found	First Blood	Total
Challenge 4	7 661	16	25 180

### 4.8.1 Identification

The general approach was to use `hashcat --identify Challenge4.txt` to list out all possible hash types. After a list of hash modes was obtained, we moved to hit-n-trial approach to identify the hash type by using hashcat dictionary attack mode with `rockyou.txt` as a base dictionary. As soon as we got more than 2 plains recovered, we confirmed the hash type for the challenge. For this particular challenge the use of `SHA256(MD5($pass))` was confirmed.

### 4.8.2 Attack Approach

We kept the same attack approach, i.e. recover partially using basic wordlists and rules, analyze and proceed to advanced attacks. The pattern was identified as permutation of the word `crackthecon2022` along with other words. We restarted attacks with custom charset. `-1cCrRaAkKtThHeEoOnN?d?s` and incremental mask `?1?1?1?1?1?1?1?1?1 ?1?1?1?1` from minimum length 6.

- Attack modes used: Dictionary (with rules) + loopback, Masked brute-force
- Wordlists used: `rockyou.txt`, `crackstation.txt`
- Rules used: `pantagrule.one.royce.rule`, `T0X1cv1.rule`, `leetspeak.rule`, `nsa64.rule`
- Mask generator used: Manual

### 4.8.3 Learning Points

Improvement in analysis is needed. The actual plains were list of asteroids mixed with names of cars. Those “other” words.

## 4.9 Challenge 5

Hashlist	Plains Found	First Blood	Total
Challenge 5	0	0	6 454

### 4.9.1 Identification

The general approach was to use `hashcat --identify Challenge5.txt` to list out all possible hash types. After a list of hash modes was obtained, we moved to hit-n-trial approach to identify the hash type by using `hashcat` dictionary attack mode with `rockyou.txt` as a base dictionary. This ultimately failed due to the `SHA256(SHA1($PASS))` algorithm not being part of `hashcat`. The algorithm was revealed to us after the end of the contest by the participating staff.

### 4.9.2 Learning Points

Don't limit testing to a single set of software. If one tool exhausts, try similar tools and tinker with their available options.

## 4.10 Challenge 6

Hashlist	Plains Found	First Blood	Total
Challenge 6	159	61	43 609

### 4.10.1 Identification

We did not try to recognize the hash types for this challenge until late in the challenge. The tweet by organizers hinted this challenge contains 5 types of hashes. With approach similar to `Challenge1`, we were able to successfully identify all the hash types.

- `md5($salt.md5($pass))`
- `md5($salt.$pass.$salt)`
- `md5(md5($pass).md5($salt))`
- `md5($salt.md5($salt.$pass))`
- `md5($salt.md5($pass.$salt))`

#### 4.10.2 Attack Approach

Since we started this challenge later in the event, much analysis was not done and attacks only comprising of basic wordlists and rulesets were done.

- Attack modes used: Dictionary (with rules) + loopback, Masked brute-force
- Wordlists used: `rockyou.txt`, `crackstation.txt`
- Rules used: `pantagrule.one.royce.rule`, `T0X1cv1.rule`, `leetspeak.rule`, `nsa64.rule`

### 4.11 Challenge 7

Hashlist	Plains Found	First Blood	Total
Challenge 7	122 999	12 554	239 127

#### 4.11.1 Identification

This challenge had a password protected ZIP file. The password was fairly non complex and was recovered with basic wordlist. Upon extracting the contents of the zip file, a hash file was provided. Applying identification techniques from Challenge1, the hash was straight up identified as NTLM.

ZIP Password recovered: `nopassword`

#### 4.11.2 Attack Approach

We kept the same attack approach, i.e. recover partially using basic wordlists and rules, analyze and proceed to advanced attacks. The pattern was identified to be list of names of fungi with some common names.

- Attack modes used: Dictionary (with rules) + loopback, Masked brute-force
- Wordlists used: `rockyou.txt`, `crackstation.txt`, `facebook-firstnames.txt`
- Rules used: `pantagrule.one.royce.rule`, `T0X1cv1.rule`, `leetspeak.rule`, `nsa64.rule`

#### 4.11.3 Learning Points

For fast hashes, such as NTLM, if wordlists are small, make sure to supply enough rules to keep the GPUs fed up with works. A large workspace is to be created.

## 4.12 Challenge 8

Hashlist	Plains Found	First Blood	Total
Challenge 8	19 277	25	?

### 4.12.1 Identification

This challenge had a password protected RAR file. The password was fairly non complex and was recovered with basic wordlist. Upon extracting the contents of the zip file, a hash file was provided. Applying identification techniques from Challenge1, the hash was straight up identified as SHA2-512.

RAR Password recovered: 123456

### 4.12.2 Attack Approach

We kept the same attack approach, i.e. recover partially using basic wordlists and rules, analyze and proceed to advanced attacks.

- Attack modes used: Dictionary (with rules) + loopback, Masked brute-force
- Wordlists used: rockyou.txt, crackstation.txt
- Rules used: pantagrule.one.royce.rule, T0X1cv1.rule, leetspeak.rule, nsa64.rule

## 4.13 Challenge 9

Hashlist	Plains Found	First Blood	Total
Challenge 9	188	74	12 180

### 4.13.1 Identification

With the same approach as Challenge1 and the hash type was identified. Apache \$apr1\$ MD5, md5apr1, MD5 (APR).

### 4.13.2 Attack Approach

We kept the same attack approach, i.e. recover partially using basic wordlists and rules, analyze and proceed to advanced attacks. The pattern was identified as permutation of the word crackthecon2022. We restarted attacks with custom charset -1 cCrRaAkKtThHeEoOn?d?s and incremental mask ?1?1?1?1?1?1?1?1?1?1?1?1?1 from minimum length of 6.

- Attack modes used: Dictionary (with rules) + loopback, Masked brute-force
- Wordlists used: rockyou.txt, crackstation.txt
- Rules used: pantagrule.one.royce.rule, T0X1cv1.rule, leetspeak.rule, nsa64.rule
- Mask generator: Manual

## 4.14 Review from the Pro Team

As the Pro and Street teams operated independently, and the street team for the first time in any capacity - the incentive was formed to provide constructive feedback to the Street team. To this end two members of the Pro team (Vavaldi and penguinkeeper) looked at the performance of the Street team and analyzed how their attacks were performed and how they could have been improved. Although the street team received this feedback in full. This writeup does not only seek to inform but also educate, and is therefore going to share some of the points made by them on how their performance could be improved.

Initially the consensus was that the Street team could have been more active and communicative with few of the available discord channels having been used to discuss the lists and progress and we wish that it was possible to include one or two pro players within the Street team to help guide them in the right direction. The most important feedback is that rockyou and best64 simply aren't the best ways to attack a wordlist anymore, perhaps in 2009 but not in 2022. To that end are included two short writeups from Vavaldi on how he was able to beat the Street team in a short timeframe showing the attacks ran. This is taken out of context of the discord so you'll have to read between the lines a little :) . This was performed on a single 3070 for hardware reference, so a team with less hardware could have performed similar attacks in double the time and still fall within the contests' timeframe. Though of course point priority, algorithm identification time is not factored in. If users have questions about these writeups you are free to email vavaldi@hashmob.net.

637970686572636F6E

You achieved 46 973 founds.

- HashMob.net Medium found wordlist attack with fordy 50k and -loopback gives us +33 535 and loopback gives us 44 258 total in <5 minutes.
- Empty wordlist + \_nsa dive v2 with loopback gives us +2 029 for a total of 46 287 in <1 minute
- Empty wordlist + Robot currentbestrules with loopback gives us +2 029 for a total of 50 767 in about 2 minutes
- Hashmob Combined full found wordlist with best64 and loopback gives us +2 644 for a total of 53 411 in 4 minutes
- An iteration of empty wordlist with loopback and -g 10k gives us +1 667
- Empty Wordlist + RuleToRuleThemAll with loopback gives us +743 for a total of +55 821 in <30s

*I'll leave it there. I think that choosing the right wordlists and just spamming rules at it for a few minutes could've given you a large initial coverage of the plains (69.80%). Remember that HashMob has a lot of resources and guides available from people who have done this for years so utilizing them correctly is often key to getting a good start.*

The above showcases how any team could have achieved around 56 000 founds in relatively short time. This was a score only 9 out of 18 teams achieved, meaning that with 15 minutes

of runtime you could be in the top 50% when effectively building your attacks.

## Challenge 1

This challenge was slightly more difficult with the Street team already performing well in this category relative to others - yet we still wanted to show how we would tackle this list from a zero knowledge perspective.

Vavaldi Today at 4:18 PM

You achieved 11 721 founds

HashMob.net Large + Fordyv2 rules + loopback gives 5 572 and with loopback 6 412 . Since I used a large wordlist with a lot of rules it took 42 minutes

Alternatively I could do this for a faster runtime (which is generally better)

```
Had I done HashMob medium + Fordy50k + loopback it would have given me `2 970` in about 2 minutes
An empty wordlist + rule stacking top_250 and fordys50k would give `+1781` for `4 751` total in about 7 minutes
An empty wordlist + Fordyv2 rules gives `+368` for `5 119` total in <1 minute
An empty wordlist + Robot_CurrentBestRules gives `+101` for `5 220` in <2 minutes
Brute force on -a3 ?a?a?a?a?a?a -i resulted in `+110` for `5 330` total in <2 minutes
Brute force on -a3 ?a?l?l?l?l?l?a?a -i resulted in `+110` for `5 330` total in <2 minutes
Which would bring you to a similar point as above in a shorter timeframe but with different founds. You can then run another hashmob large with small ruleset to grab a few extra base words
```

An empty wordlist + Robot\_CurrentBestRules gives +165 within 2 minutes

An empty wordlist + \_NSAKEY dive v2 gives +161 within 1 minutes

Noticing a lot of numbers in the founds so trying to do some rule stacking with ?d up to ?d?d?d?a append rules generated with mp64 (maskprocessor in hashcat utils)

An Empty wordlist + fordys10k + short\_append\_1-4digit +71 within 1 minute, limited effectiveness ( 6809 total)

An Empty wordlist + best64 + best64 + best64 +95 within 1 minute, limited effectiveness ( 6906 total)

Looking through the newest founds and looking up their founds without special char or numbers I get a few hits on bacteria so construct a list around it. For this I used <https://www.itis.gov/downloads/index.html> but any taxonomy or similar data will work, just make sure that the 2-3 hits you found are PART OF THE DATASET. I downloaded the MySQL Bulk so that I could parse the data from the insert queries. I kept original casing so that the I rule could do its magic.

While it's extracting and downloading I run  
 Brute force on -a3 ?a?!?!?!?!?!-i resulted in +42 for 6 948 total in <2 minutes  
 Running the new wordlist with best64 gives +20 new hits. Those hits show that it's mostly focused on the species and not family names so I duplicate the wordlist, replace spaces with newlines and run things again with more rules  
 New wordlist + fordyv2 and loopback gives +2 703 for 9 671 total in 2 minutes or so  
 New wordlist + Robot\_currentbestrule and loopback gives +676 for 10 347 total in 2 minutes or so

Because most founds are lowercase and I didn't lowercase everything I don't get all the hits since it requires the l rule most of the time or a T0 so rule comboing or changing my dataset will help. First:  
 New wordlist + rule stacking top\_1500 and top\_1500 and loopback gives +1 443 and +1 897 after loopback for 12 244 total in 10 minutes or so

Let's make all words lowercase and run fordyv2 and robot\_currentbestrule again to get another +345 and +74 founds. Alternatively you can make a file with a single l rule in it and stack that. New total is 12 663. Running again took about 5 minutes

Getting lower on founds so let's analyze a bit more. we use gramify with character (kgram) splitting on our founds with the command `python3 gramify.py character bacteria.txt bacteria --min-length=4 --max-length=128`. We then grab the `k_start` and `k_end`, sort it with the suggested command and throw it in the combinator.exe with -r fordyv2. which looks like:

```
combinator.exe k_start_bacteria_sorted.txt k_end_bacteria_sorted.txt | hashcat -m2600 -O
Challenge1.txt -o challenge_1.new --potfile-path=challenge_1.found -r
D:\Rules\Fordyv2.rule
```

This attack can be replaced for a smaller rule list to run it faster. Total runtime was 30 minutes before it was stopped early (meaning you can get more letting it run longer, avg 36 per minute).

This particular attack uses common prefixes, suffixes and rules to generate a powerful combo. Giving us another +1 390 founds for a total of 14 053. This is the 4th most in the contest and this attack took about 5 minutes.

Most of the newest plains often include a ?d or ?s in the prefix so let's make some prefix rules for that, 1 character. They also often include a digit in the center (mostly 0's) so let's create an i and s rule as well so we can stack them together. maskprocessor mp32.exe from hashcat-utils can help us with this `.\mp32.exe -1 ?s?d -2 si -3?dABCDE "^^?1 ?2?3?d > bacteria.rule"`. We can now run bacteria.txt (our generated bacteria data) with these rules to see if we can get some more and include loopback. Example of the command below:

```
hashcat -m2600 -O Challenge1.txt -o challenge_1.new --potfile-path=challenge_1.found
bacteria.txt -r rules\best64.rule -r bacteria.rule --loopback
```

This gives us +376 founds after loopback for a total of 14 430

Doing the same with top\_250 instead of best64 gives us another +354 founds after loopback with 14 784 founds total. To finish this off we'll do one more pass of empty.txt (empty wordlist) with some different rule files like fordybigboy, robotbest, nsa\_dive, and reduce our previous bacteria.rule to only prefix and not insert/replace and stacking that in turn with the other rules we've used before we can get another summed +1 669 for a total of 16 453 (inserts and replaces were slowing us down a lot) (all these last rule attacks with empty.txt combined took < 10 minutes)

To let randomness do some talking we can do empty.txt with -g 1000000 a few times to get another +397 for a total 16 850. (edited)



These random ones have added a bunch of Prefix + toggles so let's combo some forty10k with prefix and toggles. This gives us another +719 for a final 17 604 and we can keep combining this with even more rules, iterations, do some more rules and keep repeating the process as much as possible until we only get a few hits and then we'd continue to look for another patterns we're potentially missing and so on. The last attack also brought up a lot of 'crackthecon' founds so we can run a few more attacks I performed above with rules etc. I won't document the next few attacks but they're basically just rule attacks on empty.txt. We end up with a final of 17 808.

The primary thing here is realizing that themes are often common and once you run dry on the basic stuff you can try to detect or search for base words that were used. Since we're attacking a fast algorithm like md5 or md5x2 or sha1 we can run fast short attacks which gives us a lot of coverage and diversity. If we were to add debug rules we can re-use our debug rules when searching for new base-words and have a good chance of discovering new founds. (edited)

The above write-up of Challenge 1 showcases how you can perform basic attacks, analyze, perform different attacks based on your analysis, analyze and perform more attacks. Each time you see your founds going down it's an indication that you're exhausting what you can recover with what you currently have available. This means that you need to start introducing some form of variation to try and exploit a new pattern. With this write-up a team could have achieved around 17 800 founds. This would put them in the third spot of teams on that list after having spend approximately 4 hours spent on it alone. With a team working on it with more than one graphics card this can easily be done in half the required time and can likely get even more.

## Chapter 5

# Closing Notes

This concludes the write-up of both the Pro and Street team of HashMob.net. Both teams have had many learning opportunities and were shown room for improvement in different forms. We again wish to thank the contributions of all members of both teams, and the contest staff. This write-up was written not to only talk about how we achieved what we did, but also how we could have improved ourselves even more. We discussed the things we struggled with and hope you learn both from our successes and mistakes in your future endeavors. 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). It's 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 skill sets and most relevant questions can be answered expertly. To finish up the write-up we'll leave you with a small list of things that went wrong to humor you.

### Issues

- HashMob.net upload limit was still in place, limiting 5MB & 2 lists per hour.
- HashMob.net submissions could not be > hashcat mode 9998 (instead of 99998)
- HashMob.net verification prevented the uploading of md5x4 and above.
- Hashtopolis creating 21,000 subtasks for an attack with rules.
- Hashtopolis pushing too many founds to the Pro Team's instance resulting in 10+ pages of backlog.
- Hashtopolis reporting negative keyspaces processed / progress.
- Making assumptions (like assuming everyone else had already checked 2-3 words for the Troubador list).

Though most issues were resolved in their own time, a contest always pushes the boundaries of what is possible.