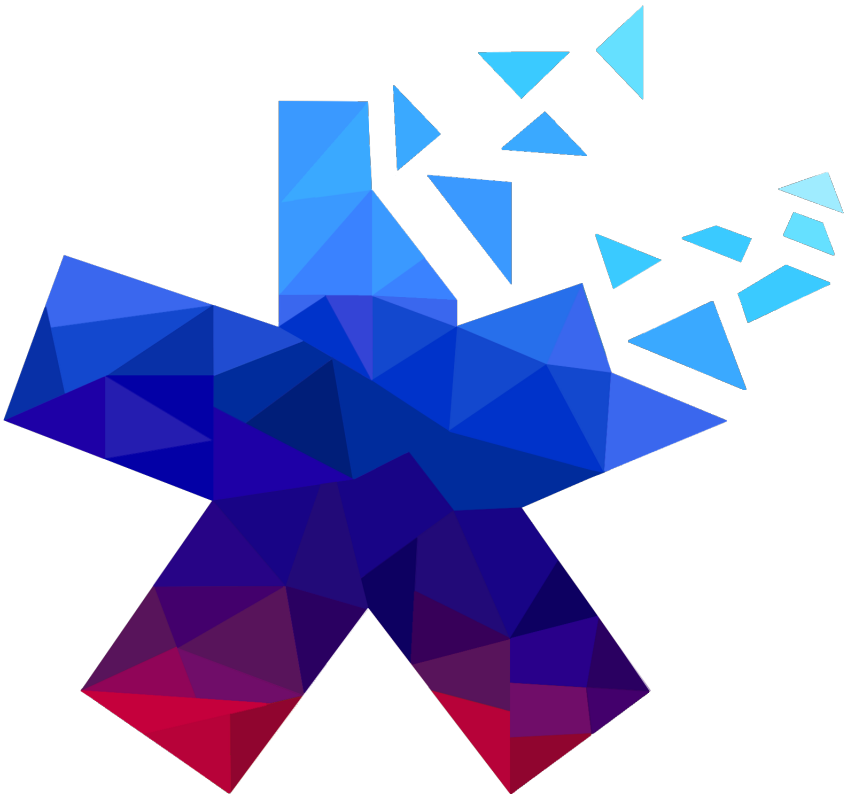
HashMob

Z2lzdDpjZGFjMWM4ZGIxZTA5NGYz
NWZmYzZjMWY2YWZiZWUwMA==

What's this?
Free to collaborate when stuck



CrackMeIfYouCan 2024
Write-up
HashMob.net
vavaldi@hashmob.net

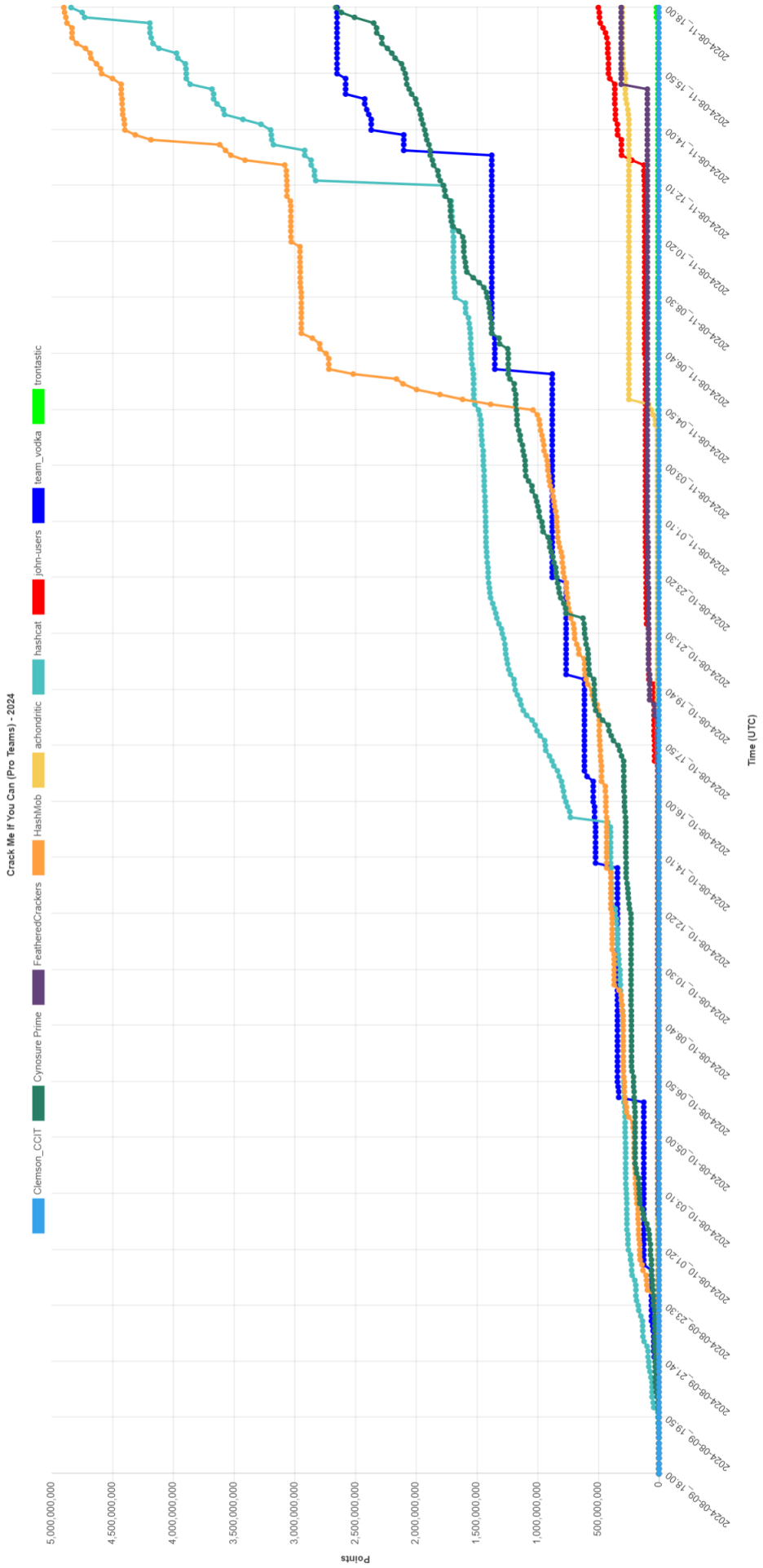
Chapter 1

Preface

Allow me to preface this write-up by beginning to thank the organizers: KoreLogic, for organizing the contest as well as the amazing members of HashMob.net who contributed to the team's performance during the competition. HashMob participated with one team this year, instead of two different teams in previous years. Our street team had more members than KoreLogic would like and had a large victory last year. To prevent it from happening again, we decided to combine both teams into one.

The Pro team achieved their second victory in a competition (following the Victory in CTC 2022), attaining the #1 spot on the CMIYC leaderboard. Slightly ahead of team Hashcat who submitted their final founds in the last seconds of the contest and nearly caught up to us, leading to a thrilling photo finish. Thanks Cynosure Prime for putting up a good fight, and great work to the new team_vodka who briefly stole the show early. We look forward to participating in more contests in the future and hope to see you all there.

Team	Points	Last Change (UTC)	bkr256	bcr256	shiro2	bcrypt	sm3crypt	saph512	rc2	adsync	radmin3	striphash	nt	Class
HashMob	4,898,862,854	2024-08-11 17:59:34	15	18	765	137	207	134	391	296	66	740	2,262	Pro
hashcat	4,838,953,118	2024-08-11 17:59:53	6	5	359	611	484	171	252	368	52	1,297	2,042	Pro
Cynosure Prime	2,659,701,506	2024-08-11 17:58:49	3	28	63	313	516	369	9	571	51	708	1,976	Pro
team_vodka	2,650,242,314	2024-08-11 15:44:20	6	6	356	180	7	133	6	298	15	442	857	Pro
john-users	498,370,079	2024-08-11 17:59:33	6	7	14	34	50	61	18	68	24	1,228	1,335	Pro
FeatheredCrackers	310,651,255	2024-08-11 17:39:49	6	5	0	25	7	13	6	77	7	191	480	Pro
achondritic	301,882,622	2024-08-11 16:42:11	6	7	0	14	11	7	11	105	8	254	789	Pro
trontastic	19,806,217	2024-08-11 17:58:35	0	0	0	1	0	7	0	112	5	321	973	Pro
Clemson_CCIT	767,545	2024-08-11 17:59:28	0	0	0	0	0	0	0	0	10	206	453	Pro


Crack Me If You Can (Pro Teams) - 2024
trontastic
team_vodka
john-users
hashcat
achondritic
HashMob
FeatheredCrackers
Cynosure Prime
Clemson_CCIT
Points
5,000,000,000
4,500,000,000
4,000,000,000
3,500,000,000
3,000,000,000
2,500,000,000
2,000,000,000
1,500,000,000
1,000,000,000
500,000,000
0
Time (UTC)
2024-08-11_18:00
2024-08-11_15:50
2024-08-11_14:00
2024-08-11_12:10
2024-08-11_10:20
2024-08-11_08:30
2024-08-11_06:40
2024-08-11_04:50
2024-08-11_03:00
2024-08-11_01:10
2024-08-10_23:20
2024-08-10_21:30
2024-08-10_19:40
2024-08-10_17:50
2024-08-10_16:00
2024-08-10_14:10
2024-08-10_12:20
2024-08-10_10:30
2024-08-10_08:40
2024-08-10_06:50
2024-08-10_05:00
2024-08-10_03:10
2024-08-10_01:20
2024-08-09_23:30
2024-08-09_21:40
2024-08-09_19:50
2024-08-09_18:00

Pro Team

- Vavaldi
- penguinkeeper
- Shooter3k
- Aaron-T
- AdamBlack
- Brad
- Cake
- Cochino
- Coin
- DAK
- outwrest
- WHYPHY
- _0.0.0.0_
- _cin
- afsa
- clem9669
- cyclone
- Flagg
- gatete
- justpretending
- kpd
- lapsikmees
- mostwanted002
- stumblebot
- w00dsman
- TalentedGuy

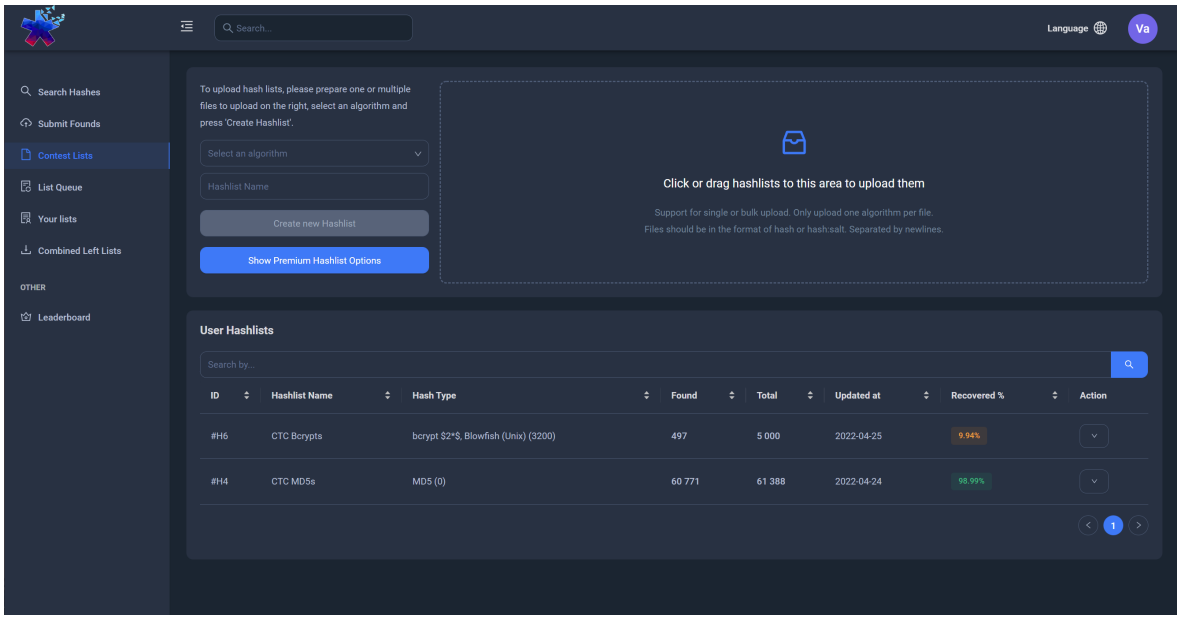
1.1 About HashMob

HashMob is a large, mostly discord-based, community that 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. We spend a lot of time working with cracking hashes and performing research on passwords. HashMob was founded in 2021, almost half a year after Hashes.org closed its doors in January of 2021. Since then it has recovered over 682 066 653 new passwords / plaintexts and amassed a following of nearly 4000 members.

1.2 Contest Environments

Since we were merging the teams, we decided to reorganize our infrastructure and setup as well. We implemented a WireGuard VPN inside a DMZ protected by a PFSense firewall. Team Members could request access to the VPN, which granted them access to our internal environment. From the internal environment it was possible to connect to a fileshare (Samba installed on Ubuntu), our tools, and our other services.

For managing hashlists we used a clone of the HashMob website again, tweaked to automatically submit new finds to CMIYC.



We had a hashtopolis environment with some custom adjustments where we tried to deploy larger attacks that would net us more insight or points. This environment allowed us to combine the power of rented hardware and everyone's personal computers to work towards completing a single larger keyspace, which Hashtopolis distributed for us automatically. We did run into some minor issues with Hashtopolis, resulting in negative keyspaces, but were able to mitigate the errors enough to continue working with it.

Chapter 2

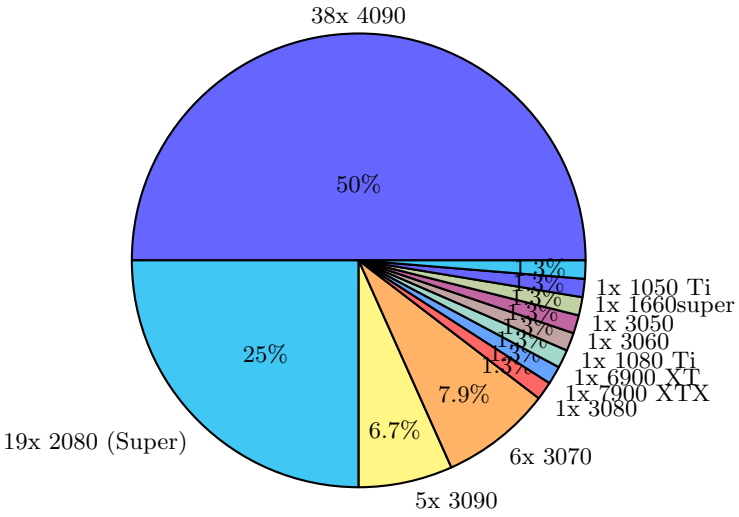
Pro Team write-up

2.1 The Preparation

In preparation for the contest the members of the Pro team actively worked on setting up the new environment and working on a few new projects for use during the contest. We leveraged the ability to distribute keyspace over many machines using Hashtopolis (HTP). This enabled us to utilize the collaborative power of many members more efficiently to perform large attacks. For example, we could rent extra power from vast.ai or other locations and hook them up to hashtopolis. They would then automatically pick up new tasks we specified and start cracking hashes for us with minimal effort.

2.2 The Hardware

The Pro team had a few great contributors to the overall power of the team with one member adding more than 12x 4090 by themselves. Roughly 32 self-owned Graphics cards were used and an additional 15+ 4090 were rented for short bursts or to work through specific attacks via Hashtopolis.



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 (say: archives) KoreLogic throws at us.

Overview of Used Software			
Name	**Open Source**	**Public**	**Purpose**
autocrack	no	no	Auto Cracking
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
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 Hashtopolis instance was modified to automatically submit any founds back 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). We created a custom version of hashcat that adds and adjusts some functionality. For example: it introduces a configuration file that presets parameters such as parameters for hashcat brain. Additionally, we added a new parameter to hashcat: -H<id>/ −−hashmob-id <id> which was a single parameter that set several other parameters for our sync script (potfile path, hashlist file, output file, and some others). The sync script would help automatically download left and found lists, as well as upload new cracks and had the purpose of keeping everyone in sync with each other.

2.3 Ready? Set...

Go! The contest started with two files being provided to us: "cmiyc-2024_pro_passwd_1.pgp", and "cmiyc-2024_pro_files_1.tar.pgp". The passwd file contained a large assortment of usernames and 'random' hashtypes. These hashes we classified into the following types:

Hashlist	Hashes Found	Hashes Total
NTLM	2 262	3 517
SHA1-CUT	740	912
Radmin3*	66	248
ADSync (MS-AzureSync)	296	2 527
RC2	391	3 245
SAPh512 (X-isSHA512)	134	2 553
sm3crypt (SM3)	207	3 331
bcrypt	137	3 444
shiro2	765	3 379
bcr256 (12-cost)	18	2 738
bkr256 ($2k$)	15	2 599

* We did not receive the Radmin3 hashes until later from the radmin.reg.bz2 file inside the cmiyc-2024_pro_files_1 archive.

2.4 Progression

As the contest progressed from one puzzle to the next, KoreLogic released more files. In total 6 cmiyc-2024_pro_files were released. Each containing more information or hints about the plains and we will go into more detail on these later. HashMob finished in first place with a total of 4 898 862 854 points. Just slightly ahead of team Hashcat with 4 838 953 118.

Points were assigned based on cracks. KoreLogic used a point-per-hashtype system that granted a set amount of points per crack per algorithm. There was no application of first-blood, removing any time pressure constraints on submission.

Hashlist	**Points per Hash**
NTLM	1
SHA1-CUT	337
Radmin3*	69 767
ADSync (MS-AzureSync)	102 535
RC2	360 449
SAPh512 (X-isSHA512)	524 287
sm3crypt (SM3)	557 057
bcrypt	4 194 303
shiro2	4 456 448
bcr256 (12-cost)	16 760 831
bkr256 ($2k$)	16 777 215

Chapter 3

Analysis

Our general approach to attacking the lists was trying to crack a few of the hashes using a mix of common passwords and specialized lists we harvested after analyzing the founds; attempting to identify possible sources. For example: we identified a large amount of Mass Effect related content (Game) in one instance and began scraping several websites to create our own (n-gram'd) wordlist.

The following rules are the ones we used most during the contest (based on our rule aggregation tool):

```
'6 d ]
$1 d ]
'7 d ]
*69 d ]
T0
c p2 i41 i92
$! E '8
'6 d D0
$!
sa4 sg6
$5
$  x0B
$1 d 'B
c
E p2 'B
i6  E
```

The most common rule-components we used (based on our rule aggregation tool) were:

```
$1
$0
$2
c
]
$!
E
d
$3
$9
$7
$5
$8
$6
$@
$4
u
D9
[
```

```
T0
'6
$*
'7
D1
t
'8
D3
D2
^A
D8
}
^i
$$
p2
D7
'9
$#
```

3.1 Archives

3.1.1 cmiyc-2024_pro_files_1

The first archive was released at the start of the contest and contained 4 files. "arj.tgz", "bundle.zip", "gocryptfs.tar.bz2", "radmin.reg.bz2" and a README file. The README file contained the instruction that file hashes were not worth points, and therefore were not worth submitting.

The radmin.reg.bz2 contained the radmin3 hashes previously mentioned. We extracted the .bz2 archive using your standard tools (winrar, bzip2). We extracted the radmin hashes by following this tutorial: `https://www.synacktiv.com/en/publications/cracking-radmin-server-3-passwords` and attempted to crack hashes using the following program: `https://github.com/synacktiv/Radmin3-Password-Cracker`.

The gocryptfs.tar.bz2 contained a slew of GoCryptFS containers, an algorithm we previously saw in 2022 and approached it in almost exactly the same way. Right after the contest we managed to successfully write an improved cracker for gocryptfs that went at approximately 60 H/s on a Ryzen 7 3700X w/ 16 threads. The general validation flow of our cracking attempts was:

```
cipher = AES-GCM using HKDF(scrypt(password,salt,N,R,P,Len))
try cipher.decrypt(EncryptedKey):
  if pass:
    return Cracked
```

We attempted to crack the password protected bundle.zip, but were eventually unable to do so. The trick to this archive was that each file was encrypted with a small, but unique password. The combined password for each file individually would make up the final document password. However, john's tools only extract the hash of the first archive. Meaning that this was uncrackable if you didn't properly extract all hashes for the archive.

The arj.tgz file contained a large amount of .arj (Archived by Robert Jung) archives. We

attempted to crack them using an old ARJ cracker: `http://old-dos.ru/files/file_1677.html`. First in windows, (see screenshot), and then in linux. This allowed us to open the archive and view its contents. We continued cracking the archives throughout the contest and later discovered some interesting correlations between the contents of the notes and the users (we will get into later).

```
➜ mostwanted002@jormungandr Downloads ./YAAC -pENGLISH.DEF -l0 -g16 ../Documents/CMIYC-2024/Hashlists/cmiyc-2024_pro_files_1/arj/abethea.arj
 YAAC (Yet Another ARJ Cracker) v. 1.0a
 Copyright (c) 1994-2001 PSW-soft by P.Semjanov
 http://www.password-crackers.com/
 (c) PSW-soft Password Cracking Library PCL v. 2.0c by P. Semjanov

Processing archive: ../Documents/CMIYC-2024/Hashlists/cmiyc-2024_pro_files_1/arj/abethea.arj, version: >2.70
Testing note_lakenyam, version >2.70
The password may begin with:
Oa, Ob, Oc, Od, Oe, Of, Og, Oh, Oi, Oj, Ok, Ol, Om, On, Oo, Op, Oq, Or, Os, Ot, Ou, Ov, Ow, Ox, Oy, Oz, OA, OB, OC, OD, OE, OF, OG, OH, OI
, OJ, OK, OL, OM, ON, OO, OP, OQ, OR, OS, OT, OU, OV, OW, OX, OY, OZ, O0, O1, O2, O3, O4, O5, O6, O7, O8, O9, O{, O}, O:, O", O<, O>, O?,
O[, O], O;, O', O,, O., O/, O~, O!, O@, O#, O$, O%, O^, O&, O*, O(, O), O_, O+, O`, O-, O=, O\, O|, Pa, Pb, Pc, Pd, Pe, Pf, Pg, Ph, Pi, Pj
, Pk, Pl, Pm, Pn, Po, Pp, Pq, Pr, Ps, Pt, Pu, Pv, Pw, Px, Py, Pz, PA, PB, PC, PD, PE, PF, PG, PH, PI, PJ, PK, PL, PM, PN, PO, PP, PQ, PR,
PS, PT, PU, PV, PW, PX, PY, PZ, P0, P1, P2, P3, P4, P5, P6, P7, P8, P9, P{, P}, P:, P", P<, P>, P?, P[, P], P;, P', P,, P., P/, P~, P!, P@
, P#, P$, P%, P^, P&, P*, P(, P), P_, P+, P`, P-, P=, P\, P|, Qa, Qb, Qc, Qd, Qe, Qf, Qg, Qh, Qi, Qj, Qk, Ql, Qm, Qn, Qo, Qp, Qq, Qr, Qs,
Qt, Qu, Qv, Qw, Qx, Qy, Qz, QA, QB, QC, QD, QE, QF, QG, QH, QI, QJ, QK, QL, QM, QN, QO, QP, QQ, QR, QS, QT, QU, QV, QW, QX, QY, QZ, Q0, Q1
, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q{, Q}, Q:, Q", Q<, Q>, Q?, Q[, Q], Q;, Q', Q,, Q., Q/, Q~, Q!, Q@, Q#, Q$, Q%, Q^, Q&, Q*, Q(, Q), Q_,
Q+, Q`, Q-, Q=, Q\, Q|, Ra, Rb, Rc, Rd, Re, Rf, Rg, Rh, Ri, Rj, Rk, Rl, Rm, Rn, Ro, Rp, Rq, Rr, Rs, Rt, Ru, Rv, Rw, RW, RX, RY, RZ, R{, R}
, R[, R], R~, R^, R_, R`, R\, R|,
The 3rd char is:
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, B, C, D, E, F, W, X, Y, Z, 7, 8, 9, {, }, :, <, >, ?, [,
], ;, ~, @, ^, _, `, =, \, |,
Recommended set of chars : [ $a $A $1 $! ]

Processing line 56 of password definition file...
Testing 0-chars passwords ...
Testing 1-chars passwords ...
Testing 2-chars passwords ...
Testing 3-chars passwords ...
Testing 4-chars passwords ...

Oa~z  CRC error!
Obfi  CRC error!
Odfi
Offi
Offi - CRC OK
In hex (PCL style): \4F \66 \66 \69
Passwords tested = 78875 (time = 00.01, rate = ***** p/s)
Totally   tested = 97058, slow = 90264
```

```
>arj cat note_lakenyam
Ne
```

3.1.2 cmiyc-2024_pro_files_2

The second archive contained 4 files. alg_issue.txt, 2 flavor text leak_ files, and one pro.pcap file. We were looking for hints in the leak_astroturf and leak_drone files at first, but at the end of the contest discovered these were simply flavor text. The alg_issue.txt file seems to make references to the RC2 list in that it's "nothing off the shelf". Probably requiring a hint from KoreLogic, or for us to properly analyze the content and find possible modifications (such as converting base64 to hex, or transposing the hash).

We were able to extract the pro.pcap wifi hashes using hcxtools and were able to successfully recover the password quickly. The password: base64_decode($TWljaGFlbCEh$) did not lead us anywhere else, unfortunately. According to KoreLogic, it was possible to use the Wi-Fi password to decrypt traffic that transferred credentials over HTTP/FTP. This could allow us, as attackers to try the passwords against other lists.

3.1.3 cmiyc-2024_pro_files_3

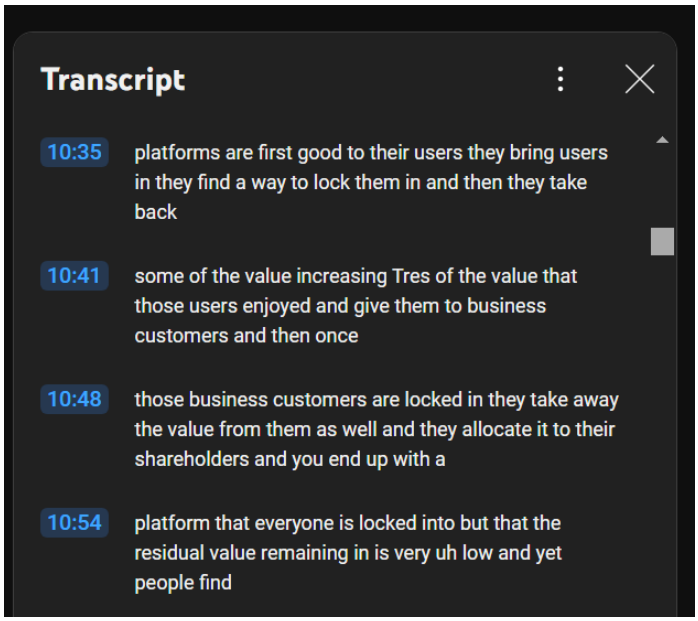
Contained two files. A "winoneforthe.zip" file containing many .zip archives. And the "maybe_sell_shoes" file Containing a malformed email chain / several messages. The winoneforthe.zip archive was protected by the password: base64_decode(*emlwcGVy*). The malformed messages contained a reference to a Java hashing library that's been mangling data for weeks. The email was mangled by removing the first letter of a word if it was a vowel (aeiou). The mangling is not something we were able to recognize or exploit in any of the lists.

```
Kelly
 hate this
junk ffshored software that  have
to make work
just shoot me

Zemus
Huh?

Kelly
h just had the most garbage Java hashing library dumped n me
pparently been mangling data for weeks
'd show you but then you'd go blind too nd who will drive me
to the bar
```

The winoneforthe.zip archive contained many zip archives where the filenames were made up of 2 names (first name + last name). We used hashcat mode 17225 to crack the files and were able to successfully recover the plaintext of 645 / 964 archive hashes. When each archive was unlocked, the file "<username>.txt" was shown, based on first character of first name + last name. Turning AgnusSalas into asalas. The ".txt" file itself contained a reference to a youtube video (based on ?v in a youtube link) + several timestamps. The timestamps refer (for youtube videos) to the time in MM:HH plus the n-th word. For 10:48:6 it refers to the 6th word shown in the transcript of the video at 10:48.



```
transcript_212RSgT0vdk
asalas-a: 10:48:6 40:34:4 49:10:5 16:23:5 13:23:12

which translates to:
asalas-a:in it guess batteries all
```

We also identified similar patterns in other hashlists, so this was a pretty big reveal for us. We found out at the end of the contest that this is not just timestamps, but also book ciphers and similar applications to movies or youtube videos. We did not figure out this nuance until later and were only able to partially make use of the information.

```
mwaldrop2:$NT$14d98d5f9df17f8ace50e58b56a10f7d:3:4:1 0:5:1
```

3.1.4 cmiyc-2024_pro_files_4

The fourth archive contained a zip file protected by the password: base64_decode($QWxsaXN3ZTEx$). The baritubbs_data.zip archive contained an account_metadata.yaml file that specified the departments of members. A gen_rc2.py file with instructions on how the RC2 hashes were built. And two flavor text leak_ files.

3.1.5 cmiyc-2024_pro_files_5

The fifth archive contained four files protected by the password: base64_decode($bGVha3M=$). App.log contained usernames, partial plaintext passwords with the last five characters redacted, and a csum field which we figured out to be a CRC8 checksum. This enabled us to get our first bcr256 by simply guessing the password of ‘utrahuman‘ to be ‘codebreaker‘ (the same as a failed user login right before). KoreLogic validated our found which allowed us to better identify the algorithm used for bkr256 as we had a valid hash:plain combination. The ”cigo” file contained a hint indicating that leading 0’s were being stripped from a hashlist (SHA1).

```
$2k$12$iyvX.kDndyq/YiUvv7J.NPApguNMJG65lr2k7H0A7Y3d7LLc1tOS:codebreaker
```

```
BUG][Login successful][user='preciousp'][pass='Dabgmao*****'][csum='33']↓
BUG][Login successful][user='apontee'][pass='Dape*****'][csum='e2']↓
BUG][Login successful][user='billupst2'][pass='Aishan*****'][csum='89']↓
BUG][Login successful][user='eppskorek'][pass='6bo*****'][csum='d1']↓
FO][Login failed][user='codebreaker']↓
BUG][Login successful][user='utrahuman'][pass='codebr*****'][csum='2d']↓
BUG][Login successful][user='mconover4'][pass='subscri*****'][csum='49']↓
BUG][Login successful][user='edaughert'][pass='Hcyd2506*****'][csum='3e']↓
BUG][Login successful][user='kathyhamme'][pass='Ensign Lo*****'][csum='67'
BUG][Login successful][user='ilanfallon'][pass='Ammaa*****'][csum='26']↓
BUG][Login successful][user='albread'][pass='Auxilliu*****'][csum='c7']↓
BUG][Login successful][user='prewittat'][pass='kilPome*****'][csum='21']↓
FO][Login failed][user='durantkr']↓
BUG][Login successful][user='durantkr'][pass='Divij*****'][csum='0c']↓
BUG][Login successful][user='apolinarif'][pass='Akshith*****'][csum='a5']↓
BUG][Login successful][user='titusrossi'][pass='Dgpcf*****'][csum='eb']↓
BUG][Login successful][user='culpkriss'][pass='Agai6nm*****'][csum='e6']↓
BUG][Login successful][user='mestclair'][pass='Divya*****'][csum='28']↓
BUG][Login successful][user='mittleidt'][pass='Palmer1P*****'][csum='97']↓
BUG][Login successful][user='ryanbollma'][pass='Donik1*****'][csum='f7']↓
BUG][Login successful][user='pamulawill'][pass='conv*****'][csum='fe']↓
FO][Login failed][user='ricksales']↓
```

3.1.6 cmiyc-2024_pro_files_6

The 6th archive contained a email from Jacklyn to Aliza and made another reference to a book cipher. The example data they refer to is the transcript hidden in the MoryMoctorow archive. The contents of which are included below the email (mmoc.txt). The password for the archive is: base64_decode($MzI6NTI6MTYgNToyNDoz$). According to one member of our team, Cory Doctorow's sci-fi book "The Rapture of the Nerds" is a recommended read!

```
To: Aliza Andersen
From: Jaclyn Kamu
Cc: Mohana Elbaz, Boyd Mcqueen
Subject: Corporate Security Policy Violations

Aliza,

I've been notified by Audit of a breach of policy by some of your
staff. Apparently they've been getting around the "don't reuse
passwords between regular and privileged accounts" rule by tying them
together with a scheme so they only have to remember one, and breaking
other rules in the process. See the attached for example data, use
32:52:16 5:24:3.

I'm told this isn't unique to your division, but is most prevalent
there.  We are going to have to force password resets on Monday, as
well as schedule additional training.

We have on file your requests for a PAM solution; nevertheless all
violations of this policy face disciplinary action.

Thank you for your cooperation.

Jaclyn Kamu
```

```
rimtaSgGz_4 transcript
32:52:16 5:24:3 7:26:17 0:42:5 3:57:5
```

3.2 Hashlists

Some patterns we identified during the contest that applied to all hashlists were:

- 3 words with spaces
- 3 words with space + symbols
- Ending in YYYY or @YYYY
- Lower + symbol
- Word + digit + same word
- Upper lower (symbol) digit (ex: Dapodcyd@2011 — Cbeca)
- Date format: 22Aug2013
- Timestamp: 32:52:16 5:24:3
- Characters from TV/Movies pulled from IMDB/Wikipedia/Fan wikis
- Names from movies: name1name
- Names from movies: name1name2name
- Indian first name with suffix DD

- Indian first name with suffix @DD
- Indian first name with suffix YYYY
- Indian first name with suffix @YYYY
- Indian first name with suffix @DDDD (rare)

3.2.1 NT(LM)

The NTLM list was fairly straightforward as it was not a very obscure or mangled algorithm. The hashes were prefixed with $*NT*$, but not much else. The passwords were relatively normal and primarily consisted of normal passwords and ngrams with rules applied to them. Some common rules were inserting a special character after or in place of a space character. Something that is fairly difficult for hashcat to do as the 's' rule only replaces all instances.

```
0ebfcabe4f9b9ef34c683fff4a163cd4:And passes it-back
297b3c121ba92453718c4cb9b7df98e5:Reliable -communication
7cd024211fad70c6ceb2272f78568d41:A!burner in the
34227038ad9db84b4a8d895c96d3ecd3:the fallen!Rifle
9347da21424048f4f8a2e858ed0bf660:the shaft of-The
7d2865824a0855b8fb36c6dede43868a:of %The beautiful
d2eef988f94895476404ac00039ac1a8:will Ensure $that
48c72dbf221938b4295b1a33462f9944:pausing$To look
42ac64b9c3279b8a3156f29ea7cf2118:Simple &Network7
835c6a6b770ec5f3801c8eefb0cad0ad:We )control the7
f44b89c457ebb0f099d70ee5de5dcfc8:procedure +is a7
016e7a9df46327c7dcbc76c0eed83d5e:His sword%into5
11382e970c4565d0a30f166831d7cf48:The mill *house1
395423f338d986476e0347c82121c48b:has $The responsibility
e0dbcab160ff36c14a3f439b6031fae8:attached To $that
56da672088e84b8797cc27ca35b00883:a Practical+level
cbed0be65f7c07d5a96534f7a563ab80:tree trunk in=The
26bc9e2df3e161806635f6a15b97677e:thin _layer Of ice
707d66044fe8aceea1bcc0960fb88b3c:Proposed!Official
2b284e16a9b9f5bb48b8a334b3d5945f:the$Flames on the
62b1ca4ff7e7d1ff18339c6b53625e2e:lying back in*The
75fe5000b65421f0fca787971c0d83cd:alwas1hapoen
85280a608fd7ae976eaf2668a703707b:pushes!them back
```

As well as some dvorak transposed passwords based on the hint given by KoreLogic. We used the following python3 script to convert wordlists to dvorak. We are still uncertain if this is a coincidence for some of the lists.

```
qwerty = "`1234567890-=qwertyuiop[]\\asdfghjkl;'zxcvbnm,./~!@#$%^&*()_+
                                    QWERTYUIOP{}|ASDFGHJKL:\"ZXCVBNM<>?"
dvorak = "`1234567890[]',.pyfgcrl/=\\aoeuidhtns-;qjkxbmwvz~!@#$%^&*(){}\"<>
                                    PYFGCRL?+|AOEUIDHTNS_:QJKXBMWVZ"

#I'm not proud of this either
qwerty = qwerty.replace("\\", "\\\\")
dvorak = dvorak.replace("\\", "\\\\")

translation_table = str.maketrans(qwerty, dvorak)

def qwerty_to_dvorak(text):
    # Translate the text
```

```
    return text.translate(translation_table)

with open("YourFileHere.txt", "r", encoding="utf8", errors="ignore") as file:
    for line in file:
        print(qwerty_to_dvorak(line.rstrip("\r\n")))
```

```
4e900e01416c4c941c8147590f0f47c2:Camxrpb2,cb
9b75fab3d41bce1877ece6e449d72e22:Dcbemryrp1
162ddf565077f1eec7d2512e0abd2d3d:I.byn.{2
30b19ef98082de2750248c572b65de7e:Tamn.od@1100
e21981bd8e92915b1eca198f5242cd07:I0eu@yd.p1
7259dfa542770aa1f471669ed9ea7ab4:Uagoycba{82
7aec3e2fe351bb862441b3a8f98c91a6:Taxcnab@1987
84f51b9d52145077ff847a1f04af486d:IGPGTPCODBA27
8593d1a817507fed6c991cbf0266df45:Tanamxgp[01
62ac5ce1afdfc31312ad19de78eeb28f:Hacdabgmab21$
6c5627d2d35334768cb903be91478bbe:Ir,ydam@0305
12e29434d7ba656befc3a23336f233af:Hgb.@051983
16fbcd580cef086bb17243cdf91809b0:Hab.yy.01!@#
5e30745d180d1ae739e03556d88ef00a:>nchad@2804
724682f5ea9ed38a48e382459f3c7830:Cxpadcm@4321
f9a64f503c4da06d17d819794bb8ebb0:Hajtorb1121#
```

3.2.2 StripHash (SHA1)

We faced some more trouble tackling this list than expected. The SHA1 hashes were modified to remove characters from it. Hashes not being fully accurate or missing characters was not an unfamiliar topic for us. Dating back to all the way to 2014 with the release of LinkedIn on hashes.org where it turned out hashcat would validate hashes, even if they were not a full match (hashcat does not validate the first 8 bytes of a sha1 hash). KoreLogic pulled this trick on us in previous years because they like playing with their food. One of the troubles we had was matching the plains back to their original after being cracked in an efficient way so we could submit them properly.

One approach we attempted was to pre-/suffix every hash shorter than 40 characters with every hex character. A replacement that looks something like this (sorry for your eyes). Which worked well, except that the size of the hashlist became 5GB and produced a lot of invalid hashes that could not be cracked.

```
\1a\n\1b\n\1c\n\1d\n\1e\n\1f\n\10\n\11\n\12\n\13\n\14\n\15\n\16\n\17\n\18\n
\19\na\1\nb\1\nc\1\nd\1\ne\1\nf\1\n0\1\n1\1\n2\1\n3\1\n4\1\n5\1\n6\1\n7\1
\n8\1\n9\1
```

Another approach we looked into was using sha1-half, focusing on the middle portion of the hash, potentially eliminating a lot of duplicates from the hashlist. Due to varying hash lengths we never ended up fully implementing this, but I believe this is a valid angle for cracking the hashes more efficiently.

The plains within the SHA1 list were very similar to those in the NTLM hashlist. We also performed an analysis of the founds to try and identify what potential wordlists might have the best effect. We did this using the HashMob plain-finder run in a loop for each plain we recovered.

```
hashes.org-2020 138
hashes.org-combined 137
hashmob.net_2022-12-31  136
hashmob.net_2022-12-31.official 135
hashes.org-2012-2019  126
0x69BE_v1_top_1000M 119
cyclone_hk_v2 100
hashes.org-2018 91
hashmob.net_2022-12-31.huge 90
COMB_passwords  90
hashes.org-2019 87
hashkiller-dict 75
hashmob.net_2022-12-31.larger 72
found.2015  66
0x69BE_v1_top_100M  56
antipublic  51
clem9669_wordlist_medium  48
found.2016  47
hashmob.net_2022-12-31.user 46
hashmob.net_2022-12-31.large  45
hashes.org-2017 39
0x69BE_v1_top_10M 34
hashmob.net_2022-12-31.medium 33
Top102Million-probable-WPA  31
clem9669_wordlist_small 31
Top120Million-probable-WPA  31
Top31Million-probable-WPA 23
LiveJournal 22
hashkiller_2022_01_11 22
rockyou 21
ignis-1M  19
SkullSecurityComp 18
0x69BE_v1_top_1M  17
10M Popular 16
hashmob.net_2022-12-31.small  16
hashes.org-2021 15
Collection1passwords  14
DailyQuiz 14
hashmob.net_2022-12-31.tiny 13
Top65Thou-probable-WPA  12
hashmob.net_2022-12-31.mini 10
zxcvbn  9
Wiki-en-all 6
artists_nopunct 5
artists 5
Top1933-probable-WPA  4
0x69BE_v1_top_1k  4
subjects  3
0x69BE_v1_top_1000M_reduced 3
weakpass_3a 3
subjects_nopunct_nospace  2
Top66-probable-WPA  2
hashmob.net_2022-12-31.micro  2
```

3.2.3 RAdmin3

We use a conversion script to convert the radmin3 hashes to a format supported by Hashcat (-m 29200). `https://fossies.org/linux/hashcat/tools/radmin3_to_hashcat.pl`. To split it up per user we used the following two scripts:

```
#!/bin/bash

# Create a directory to store the individual files
mkdir -p split_files

# Initialize variables
file_count=0
output_file=""

# Read the input file line by line
while IFS= read -r line; do
    # If the line starts with "[HKEY_LOCAL_MACHINE", start a new file
    if [[ $line == "[HKEY_LOCAL_MACHINE"* ]]; then
        # Increment the file count
        file_count=$((file_count + 1))

        # Set the output file name
        output_file="split_files/file_$file_count.reg"

        # Write the first line (the HKEY line) to the new file
        echo "$line" > "$output_file"
    else
        # Append the line to the current file
        echo "$line" >> "$output_file"
    fi
done < "radmin.reg"
echo "Data structures have been split into individual files in the '
                                    split_files' directory."
```

```
#!/bin/bash

# Directory containing the split files
split_dir="split_files"

# Iterate over each file in the directory
for file in "$split_dir"/*; do
    # Extract the user number from the file name (e.g., file_92.reg -> 92)
    user=$(basename "$file" .reg | cut -d'_' -f2)

    # Run the Perl script and save the output to {user}.hash
    perl ~/Downloads/radmin3_to_hashcat.pl "$file" > "${user}.hash"

    echo "Generated hash for user $user in ${user}.hash"
done
echo "All files have been processed."
```

Our first pattern with this algorithm was: <word>! - essentially $! as a rule. We also identified <name>1<name> and <name>1<name>2<name> as a pattern. Due to only 248 hashes existing, it was difficult to find a lot of patterns in a limited time.

3.2.4 ADSync

These hashes were MS-AzureSync PBKDF2-HMAC-SHA256 hashes that had an additional semicolon (;) at the end. Removing them allowed hashcat to load the hashes as mode 12800. Most of the plains recovered for this algorithm were popular passwords based on common dictionaries. We appended many of the popular patterns to these dictionaries throughout the contest.

3.2.5 RC2

We were the first to submit cracks for RC2, right before the hint was dropped. This surprised even KoreLogic who thought we were smart enough to figure out the algorithm (thanks for the compliment). In reality, we planted a camera in their office in 2023, which allowed us to snoop on their custom source code.

Our first cracks were obtained by correlating passwords from the GoCryptFS and GoCryptFS Notes. We found that the RC2 password from a user was based on their GoCryptFS Password + GoCryptFS Note combined into one as indicated in the below image.

	A	B	C	D	E
	User	GoCryptFS PW	GoCryptFS Note		RC2
	snellhaze	Adhira	#ra2311	Adhira2311	$RC2$100$xvIn2YMuRGE=$kIf7lbik5v3TQ9oLryb+EGpxqr4PppTL
	montalvon	Aarohi	#hi7886	Aarohi7886	$RC2$100$Pc7fifQqFXw=$YUtxplxVo+yjNoUPNiMta4g8bSPBKnC+
	crookow	Aakash	#sh2005	Aakash2005	$RC2$100$acy+uTviQnU=$txJCgXsURJtaVY+taeOaBvk6QyeDf0bM
	jwetzel	Abeer	#r101@	Abeer101@	$RC2$100$6txdEu+Rix0=$1OcbKGK7YcfB4oXfOhl21QWeRlbo2PPa
	cumberdac	Aarya	#an@11	Aaryan@11	$RC2$100$QrcckFRvpvE=$rbqEVrtmg2pjh8vEPtlGQ30UgVAuZSPP
	joycsnell	Adarsh	#sh2208	Adarsh2208	$RC2$100$fqbktpVxQQQ=$GHE2rlC99jilB4JQg6/cURORTEaQvwyc
	mcoronado	Akshara	#ara2011	Akshara2011	$RC2$100$eCdrpqWLiyY=$RAhLPmpBKTyI8TcV/iPfMje5zHvXsfI7
	adelaidah2	Akshay	#ay777*	Akshay777*	$RC2$100$M2k62B7SGBk=$buYVM94gh0JpF+yVWfvxs2UjWZtuxfJJ
	edmondday	Ananya	#ya2911	Ananya2911	$RC2$100$Bd9dBcfV4eE=$6znOwS7vT5h8MBdUDaYHyjyST2mUCZrl
	aggilbert	Ayush	#h@424	Ayush@424	$RC2$100$4XOfvEudr7o=$HvD8+EuaLBF8taCwTGs/Ewjx11mizRUI
	velascoma	Arjun	#n@098	Arjun@098	$RC2$100$z1bTZ4PuCmE=$sfKM4o2zkuldBdVXCPpwOmmMiUBNAU
	lakenyas	Bhavya	#ya143$	Bhavya143$	$RC2$100$1YHqThKfRNQ=$nBcmvulwoQ+p4ktAd6ygsp+ERQitWQj0
	acostaja2	Charan	#an2007	Charan2007	$RC2$100$rJJDLzE/gHQ=$vU9ud+nEShD2bs98EKKJgSMiTLvvl+tN

```
$RC2$100$xvIn2YMuRGE=$kIf7lbik5v3TQ9oLryb+EGpxqr4PppTL:Adhira2311
$RC2$100$Pc7fifQqFXw=$YUtxplxVo+yjNoUPNiMta4g8bSPBKnC+:Aarohi7886
$RC2$100$acy+uTviQnU=$txJCgXsURJtaVY+taeOaBvk6QyeDf0bM:Aakash2005
$RC2$100$6txdEu+Rix0=$1OcbKGK7YcfB4oXfOhl21QWeRlbo2PPa:Abeer101@
$RC2$100$QrcckFRvpvE=$rbqEVrtmg2pjh8vEPtlGQ30UgVAuZSPP:Aaryan@11
$RC2$100$fqbktpVxQQQ=$GHE2rlC99jilB4JQg6/cURORTEaQvwyc:Adarsh2208
$RC2$100$eCdrpqWLiyY=$RAhLPmpBKTyI8TcV/iPfMje5zHvXsfI7:Akshara2011
$RC2$100$M2k62B7SGBk=$buYVM94gh0JpF+yVWfvxs2UjWZtuxfJJ:Akshay777*
$RC2$100$Bd9dBcfV4eE=$6znOwS7vT5h8MBdUDaYHyjyST2mUCZrl:Ananya2911
$RC2$100$4XOfvEudr7o=$HvD8+EuaLBF8taCwTGs/Ewjx11mizRUI:Ayush@424
$RC2$100$z1bTZ4PuCmE=$sfKM4o2zkuldBdVXCPpwOmmMiUBNAU0e:Arjun@098
$RC2$100$1YHqThKfRNQ=$nBcmvuIwoQ+p4ktAd6ygsp+ERQitWQj0:Bhavya143$
$RC2$100$rJJDLzE/gHQ=$vU9ud+nEShD2bs98EKKJgSMiTLvvl+tN:Charan2007
```

```
Timestamp: 2024-08-10_19:12:04 GMT
New cracks: 13
  Lines received: 13
  Well-formed: 13
  Repeats from previous submissions: 0
```

We continued this process for the other recovered GoCryptFS passwords. Once the source code for generating the hashes was finally released. We worked on creating a cracker for it. We did this based on the script provided by KoreLogic as we fashioned ourselves a custom cracker in Python:

```
import sys
import threading
from base64 import b64encode, b64decode
from Crypto.Cipher import ARC2
from Crypto.Util.Padding import pad

def prepare_key(password):
    return password.encode().ljust(16, b'\x00')[:16]

def rotate(strg, n):
    n = n % len(strg)
    return strg[n:] + strg[:n]

def encrypt(plaintext, key, iv, rounds):
    cipher_text = plaintext.encode() if isinstance(plaintext, str) else
                                                    plaintext
    cipher_text = pad(cipher_text, ARC2.block_size)
    for i in range(rounds):
        cipher = ARC2.new(key, ARC2.MODE_CBC, iv)
        cipher_text = cipher.encrypt(cipher_text)
        rotate_distance = cipher_text[0]
        cipher_text = rotate(cipher_text, rotate_distance)
    b64_encrypted = b64encode(cipher_text).decode()
    return b64_encrypted

def crack_hash(password, hash_line):
    hash_parts = hash_line.strip().split('$')

    if len(hash_parts) != 5 or hash_parts[0] != '':
        return  # skip invalid lines

    _, algo, rounds, iv_b64, expected_hash = hash_parts
    if algo != 'RC2':
        return  # skip invalid RC2 hashes

    iv = b64decode(iv_b64)
    rounds = int(rounds)
    key = prepare_key(password)
    encrypted = encrypt(key, key, iv, rounds)

    if encrypted == expected_hash:
        print(f"{hash_line.strip()}:{password}")

def crack_hashes(wordlist_file, hash_file):
    with open(hash_file, 'r') as f:
        hashes = f.readlines()

    with open(wordlist_file, 'r', buffering=10*1024*1024) as f:
        for line in f:
            password = line.strip()
            threads = []
            for hash_line in hashes:
```

```
                t = threading.Thread(target=crack_hash, args=(password,
                                                              hash_line))
                t.start()
                threads.append(t)

            for t in threads:
                t.join()

def main():
    if len(sys.argv) < 3:
        print("Usage: rc2_cracker.py <wordlist_file> <hash_file>")
        sys.exit(1)

    wordlist_file = sys.argv[1]
    hash_file = sys.argv[2]

    crack_hashes(wordlist_file, hash_file)

if __name__ == "__main__":
    main()
```

3.2.6 SAPh512

SAPh512 (a sha512 variation of SAP CODVN H (PWDSALTEDHASH) iSSHA-1) was unsupported by Hashcat, but luckily supported by John.

```
./john --format=saph 32.left --wordlist="YourWordlist.txt"
```

Based on the source code from John we concluded that the algorithm was likely:

SHA512x14999($plain.SHA512($plain.$salt)). To further improve cracking speed we worked on creating a custom hashcat kernel (99769 (SAP CODVN H (PWDSALTEDHASH) xissha512)) which was able to recover hashes hundreds of times faster than JTR. During the creation of the kernel we spent limited time working on the list, and even when the kernel was done we were unable to fully utilize this speed increase. As a result, most of our cracks are simple wordlist finds or suffixed with numbers/years. 8 lowercase alpha + ! or . was one of the first patterns we discovered in this list.

One thing of note is that the next large release of Hashcat will include this algorithm. This applies to SM3Crypt as well.

3.2.7 SM3Crypt

Our first cracks for SM3 crypt were made based on association attacks. By taking a cracked hash by user1 and finding another hash made by user1, we were able to guess their password with 100% accuracy. For example, our first crack was based on a previously recovered radmin3 plain.

```
$radmin3$6d0062006c0061006c006f0063006b003300*x*x:leonardo.
$sm3$BoZlIHLA6SZBo4yb$l27gNpDkvkdXG5SnOmdhruTDg1yTsfIj21Y3eLs/ux2:leonardo.

$radmin3$640065006d00610067006f006f006400*x*x:Ananya@1
$sm3$Uo9Y.hP2NBgPEUO2$NdVKNTS.AG.JBo9.9IoWLWwpiW4./2iS3Ak9Obc/.j1:Ananya@1
```

Later we were able to also apply similar logic to the data from the app.log file (containing login logs, usernames, and partial passwords + CRC8 checksum) to relatively accurately guess the password of a user (within 1-5 attempts).

```
$sm3$0kQRhjWBTKvpzYRf$eUyBssbngoO4Gos2HkD9fe3OHTSNYZVvlifSnjX8Jb2:Audia391
$sm3$0kQRhjWBTKvpzYRf$eUyBssbngoO4Gos2HkD9fe3OHTSNYZVvlifSnjX8Jb2:Audio197
$sm3$0kQRhjWBTKvpzYRf$eUyBssbngoO4Gos2HkD9fe3OHTSNYZVvlifSnjX8Jb2:AudioPor
$sm3$1xd4edButtinEB9t$pFeBbs9KbAjK.44vvufC.9E7AEBM932O.0xh/WkDDW3:Adrein23
$sm3$1xd4edButtinEB9t$pFeBbs9KbAjK.44vvufC.9E7AEBM932O.0xh/WkDDW3:Adrywall
$sm3$29xb/lMgxLBtKoWM$vFfj8n9l5NPC9ly69R5ENEG7BrrU4sSn0C/EpTzCdv5:warrenc55
$sm3$29xb/lMgxLBtKoWM$vFfj8n9l5NPC9ly69R5ENEG7BrrU4sSn0C/EpTzCdv5:warriors!
$sm3$29xb/lMgxLBtKoWM$vFfj8n9l5NPC9ly69R5ENEG7BrrU4sSn0C/EpTzCdv5:warrousse
$sm3$68dkoUqVbe1PeIB3$DSNnr9d5lH794EKDV4W89VbA2GjE/0qFTUhWvgitjr.:vip_trading
$sm3$6Hqkl0ZSivcqzfgZ$aZ.eQ18MG9fdeMXCGivsaal7c2zbf55Ow1or0PRX0z4:swingin85
$sm3$6Hqkl0ZSivcqzfgZ$aZ.eQ18MG9fdeMXCGivsaal7c2zbf55Ow1or0PRX0z4:swinton78
$sm3$81dMUUkzOzvxfqI3$xyprKHMunSffzvbAcAyKqKkegjwy9rALUmpGpdRrBZB:!Amtgbis1
$sm3$9pq6bChanoMO62EB$2SWhS3GEymjty9KHhWx8/68bTVtibjbvYGnJrzuXBfD:Dhruv12345
$sm3$9pq6bChanoMO62EB$2SWhS3GEymjty9KHhWx8/68bTVtibjbvYGnJrzuXBfD:Dhruv@2015
$sm3$/bEEVBueXmdmNm0M$mZaZuEQ.T/5bbVMsNNA85/2j96H8o3Ekh6JN7oszF7B:Althea154
$sm3$DDzOneUG0aUvz5UU$Ra3LcoUg2fNEqvnpk6VnRraYE64JFu1Obpr7sCOQOC/:Igbeti99
$sm3$DDzOneUG0aUvz5UU$Ra3LcoUg2fNEqvnpk6VnRraYE64JFu1Obpr7sCOQOC/:Igborika
$sm3$dpBGB8mDJ3UAb95T$Ny92FLyBfU23fGZgoxE25N50HMbF5Dh7ijR4kisU.t3:HgbWM5VN
$sm3$EaN.q3c8Id85zMNH$qsU/0ayqUmUksNiLYehxx8HYbDlmn0rHTkPeQrzkWG6:Ventures12
$sm3$fmM9Pf97Wi67KDEK$NERLCHW71oKmUxrGipmyf6QExQe7iMtZad2YcNUOaT5:ira.klopot
$sm3$FOiRJBNeYefmAq3e$h8cB2smz9ymwr0qeiYMv2zkerdVnqa..w9lhSDPGK.5:Ammulove1
$sm3$hFbCQ5lYWhzow3v2$/BkubXVyt9qr6ucQf1ZDjJpZuL6olXQAsBTj0qnCjt0:Akshay2703
$sm3$HtyxzdpCGv0Kjoy2$lW/dBI3hJP5t2zCGxoIgnencIyuh6LCxZqfDd8Z6RT1:uAdC3N9b
```

```
Timestamp: 2024-08-11_13:36:54 GMT
New cracks: 25
  Lines received: 66
  Well-formed: 66
  Repeats from previous submissions: 2

Errors encountered:
  Other malformed lines or invalid plains/hashes: 39
```

Netting us a few cracks before we were able to validate them with a cracker. A few hours (2) before the end of the contest we created a cracker that was capable of cracking the provided hashes. Earlier attempts to create a cracker failed due to some library issues causing the resulting hash of a plain to be different from the CMIYC hash. Even when we knew the plaintext was correct due to our previous efforts.

```
➜ mostwanted002@jormungandr cmiyc-2024_pro_files_1 ./sm3crypt_mt3 ~/Downloads/hashcathashm
$sm3$/0bV.PrA7zD3PQyR$9hkY1CVro.pDy7oZzzFa5F/pv3jh4r/bKWnD1su0s63:Abarca1Abarca
$sm3$qbgeJLaKS.cRcRFz$vIGI4WR/1f6Jqgm83v1q5qrePSLcRkkTmPce7a77n.5:Aakar1Aakar
$sm3$enbEidxHSSCWQ9Ma$bfW4IlZ9.XtGsZIjDkwttg.AQMPphzDslebQ9GZKaZ.:Abuela1Abuela
$sm3$ggSaoM8TPk1.xlw7$NQnTLSHgE2XlaviZWq9tiJTFEjC5X84KNsoxLuHArE7:Adreek-Hu1Adreek-Hu
$sm3$ww/hJq4eJZEjs.kd$GpI4NGRz76zcM0Frc1ztPD1vmFs9u3lW5DXzMRisX60:Akary1Akary
$sm3$Y.zmbLsVYQBIsnwy$0nJPKDls226ZkF6r2.qarxzqZiAnAo66A1/3KHtVv8D:Alexandra1Alexandra
$sm3$TRfaJ2Faq9Zk9yu5$zY0zUzBW8CiKTY4y.Nf9Vqilcti3Pg9/tzfp2497E0A:Alexander1Alexander
$sm3$COd6YL2wyGG/gJ6b$1i6mKQfHQg9udRu9jt/o.887HF7OUMIk0QcHIS.Df2A:Amarie1Amarie
$sm3$ETG4xx6Usp9L6pzH$SsjNa.3lQT0gSJx5eALB.KXNrU5vPPiVlk98hKp9II1:Alzen1Alzen
$sm3$1aPn7oqMbyX7AXgd$v8iu0LJfALRUqRqhqVJEGD4CWLESfzjGd3YVgVg3wf/:Andorian1Andorian
$sm3$DRTdHwj2DOB64zK8$zNnyGFykPYMNk1o8U00vfE8mibXD/ua.xW5JbMuMhF7:Apollo1Apollo
$sm3$UmCoK89bi17iVPqS$RgjEovX9Y/nGcT3.z3A6qutnBq/vGfT5zCc08wjHiZ0:Anthwara1Anthwara
$sm3$jygLE5WsSKn.YatJ$WwcG.RCJO1SjIQMViIJstTnmVN0jJl0IfcUDSAX78/C:Appel1Appel
```

One thing of note is that the next large release of Hashcat will include this algorithm.

3.2.8 bCrypt

bCrypt was a challenging algorithm for us and we failed to identify significant patterns as we did with Shiro2. The bcrypt list contained various hashes with a cost factor of 10 (1 024 iterations) and 12 (4096) iterations. Our first attacks were based on common passwords and revealed a distinction between the two cost factors. Namely, the 10-cost hashes could be cracked, whereas the 12-cost yielded no results. Even after more varied attacks. We (correctly) deducted that the 12-cost hashes belonged to the bcr256 list, with the bkr256 starting with $2k$.

We identified a lot of basewords that existed in smaller lists like ignis1M, but running a wordlist with many rules is a difficult task for bCrypt as it's a slow algorithm. We discovered that many of the bCrypt passwords were of similar length, and all contained a special character. Our first attack was a reduction of the HashMob dictionary with 8-9 length passwords containing 's and .'s. This led to some results but was inefficient overall. We started applying rules and eventually figured out a portion of the passwords were made up of a word + numbers or @ + numbers where the first letter was capitalized in almost all cases. We therefore focused on heavier rule attacks with a more limited wordlist.

Included are some things we tried, most with little success:

```
123456789 with ?s ?s?s and ?s?s?s pulled from HM large prefixed and suffixed
7 len word + 2 digit ?d?s or ?s?d
Xxx-xxx-xxxx as digits with -
"Ilove" and "i_love_" + 2-4 meta words
Went over founds 10-13 len with ], ]], ]]]
4-5len uppcase word from found @ + 3-4 len digit
Looked at 4-5 len upper word with # or . or ! for a 2-4 digit
7-8 words : and c with #1 @1 and .ru .com !! .. . ! prepended and appended
Len 4-9 word from meta + 23! Or 23. T0
Len 4-9 word from meta with
T0 $1 d ]
T0 $2 d ]
Len 4-9 word from meta with 2 digit ends T0
Len 4-9 words from meta with 1 digit prepend and appends T0
Cmiyc usernames + 1 digit append and prepend
```

```
Cmiyc usernames + ! and . appended lower and T0
Founds with one space and two -> s ! and s ( , all lower
s .
s *
s +
s *
s ^
s -
s _
Word 4-9 with 1$$ append T0
Formats like "Tacnao@2021..." with 1-4 digits and @[0-9]+$
Founds that end in digits with ] and ] append ! . and *
Variations of love_ i_love_  i_<3_ a_ the_ this_ that_ what_ from_ some_ when_
their_ been_ as a prefix with len 4-9 and with spaces and +'s
Len 4-5 words with u and 4 digits
Len 3 word with ! and - then 4 digits (tried l u and ])
Prepend 2d + @ then word T0'd
```

Due to the -a9 mode not being usable with the Hashcat brain feature, some data about attempted candidates was likely lost on the machine being used to identify new patterns and then repeated by other team members looking to assist with the hashlist. Identifying ideal methods to distribute attempted candidates stably would likely have bolstered the recovery efforts.

3.2.9 Shiro2

Shiro2 is the algorithm that carried us to victory. A high-value algorithm. Where hashcat was able to utilize bcrypt to its fullest, we were able to make use of Shiro2 to propel us forward into the rankings. Shiro2 is essentially Argon2 with a slight tweak. To convert Shiro2 into a format supported by JohnTheRipper, use the following commands:

```
# shiro2 -> argon2
sed 's%^\$shiro2%%;s%\(t=[0-9]\+\),\(m=[0-9]\+\)%\2,\1%' 34.left

# argon2 -> shiro2
sed 's%^%$shiro2%;s%\(m=[0-9]\+\),\(t=[0-9]\+\)%\2,\1%' argon2.found
~/cmiyc_2024/john/run/john.pot | sed -e 's/^/\$shiro2/g' -e 's/m=65536,t=1/t=1
                                       ,m=65536/g'
```

We made use of both JohnTheRipper with –format=argon2-opencl, and Cyclone's very own Argon2 cracker (`https://github.com/cyclone-github/argon_cracker`). Our first cracks were simple passwords like in bcrypt.

```
$argon2id$v=19$m=65536,t=1,p=4$V0m2DAEb/MoHYABXGUzehQ$x:butterfly.
$argon2id$v=19$m=65536,t=1,p=4$jm0HA7o4vBhEulWoEs0X3A$x:snickers!
$argon2id$v=19$m=65536,t=1,p=4$yWVZlnrgUknAssQwn/7zDw$x:iloveyou!!
$argon2id$v=19$m=65536,t=1,p=4$1IBi9t6bbmNjaZP0PFQD4A$x:creative.
$argon2id$v=19$m=65536,t=1,p=4$PdIFVdFJqXTcsYiGu5HejQ$x:december!
$argon2id$v=19$m=65536,t=1,p=4$y33XH6zlKnKBKf+QQwE6Bg$x:nintendo!
$argon2id$v=19$m=65536,t=1,p=4$1lMRbMm4ecCnMaQfbJBelA$x:kristina!
$argon2id$v=19$m=65536,t=1,p=4$DXrfshMewhbGpCL4SO4XYw$x:nightmare!
$argon2id$v=19$m=65536,t=1,p=4$hM/JuhdDtezdFj4llWWgzg$x:rammstein.
$argon2id$v=19$m=65536,t=1,p=4$ylPMKsZ/TFRmlVn34bqRKA$x:carpediem.
```

We soon discovered words with similar rules to those we've seen before, and later also performed association attacks based on the hashes we cracked from other lists. Leading to cracks like these:

```
$argon2id$v=19$t=1,m=65536,p=4$+rou5ngisGmktsUDt/1A2g$x:radion-dankov
$argon2id$v=19$t=1,m=65536,p=4$z63Eks6y+3D7TOt0iWPucw$x:regina-rebina
$argon2id$v=19$t=1,m=65536,p=4$lCNH7sjuCEQsBvI6/vvW3A$x:Ashish@14
$argon2id$v=19$t=1,m=65536,p=4$R1frHNCtgN2qPbpaN7FAGQ$x:Akshaya@16
$argon2id$v=19$t=1,m=65536,p=4$ScM9ndoI5vPKFgzVe37i0w$x:Cbeca*12
$argon2id$v=19$t=1,m=65536,p=4$WOI9n1gEKTK5huyHDIjgfQ$x:Dabcod@2016
$argon2id$v=19$t=1,m=65536,p=4$FLbwjWm1NpkZuohGe75Trw$x:Subsidiaries0
```

And writing simple rules like this allowed us to specifically target founds that matched our expected patterns

```
:
c $@ $2$0$2$4
c $@ $2$0$1$0
c $@ $2$0$1$1
c $@ $2$0$1$2
c $@ $2$0$1$3
c $@ $2$0$1$4
c $@ $2$0$1$5
c $@ $2$0$1$6
c $@ $2$0$1$7
c $@ $2$0$1$8
c $@ $2$0$1$9
c $@ $2$0$2$0
c $@ $2$0$2$1
c $@ $2$0$2$2
c $@ $2$0$2$0
c $@ $1$4
c $@ $1$5
c $@ $1$6
c $@ $1$7
c $@ $1$8
c $@ $1$9
c $@ $2$0
c $@ $2$1
c $@ $2$2
c $@ $2$3
c $@ $2$4
c $!
c $0
$1 d ]
$2 d ]
```

* possible puzzle hint

A new realization hit when we saw "John Bates". Someone who also appeared in our Star Trek wordlist (created due to the other lists). This quickly lead to the discovery of Star Trek references, as well as other possible sci-fi references. Leading to cracks such as:

```
$argon2id$x$1m7IE+EO8Jkd0ngvEmFoAA$x:Tal Shiar Female #2
$argon2id$x$3zpiKojTN2NmhVtz1u1t0Q$x:Tony Cicci
$argon2id$x$AjH3JdRXpRbGpFOxVv+cSA$x:Voyager Computer
$argon2id$x$ArY4cgy/kHv8MnZRmBv8Lw$x:Trainee #1
$argon2id$x$B9KdlWstEjfQpc4bHjqx0w$x:Yang Drummer
$argon2id$x$Be+BixTbjbX282z7l1bC3g$x:William Telfer
$argon2id$x$D4gjjBrMbrzuXUNYQlhDOw$x:Zhat Vash Member
$argon2id$x$E1QGwCJbpFrzXYiucDxuqA$x:Voyager Security Officer
$argon2id$x$HK9FBz36gU+OQ43nyUbV+A$x:Xindi-Insectoid Councillor
$argon2id$x$IPkV7npgTsXxtpN8O8FJYA$x:Young Crewman
$argon2id$x$J3jRw+ceISg/6cQUK/4P+g$x:USS Titan Ensign
$argon2id$x$K3EDfrzgHcV7O4ffufTMsA$x:Tellarite Delegate
$argon2id$x$OBW41rES1BDi0UQ2lTFBoQ$x:The Hayseed
$argon2id$x$OQC2BNcqX3g/MXSXB/p+pQ$x:Weapons Pirate
$argon2id$x$Qk7woL4Q8tnF1Dm1/OCVhg$x:Xindi-Reptilian Soldier
$argon2id$x$SYAEeeCdkn+qmeqMUM0WhA$x:Transporter Officer
$argon2id$x$TT6sI4NpC7VbEARH58ftow$x:Thomas Harewood
$argon2id$x$VFHDjXwxl6UjDE5ZgO3zFg$x:Yeoman Rand
$argon2id$x$WA5ktQmRRxK1O8hL9OT7yw$x:Terminal Jockey #2
$argon2id$x$WSnCcZ6aRkJEGnSxOnZXyA$x:Vulcan Wedding Guest
```

Based on our hits in SM3Crypt we attempted to again try name1name, name1name2name, as well as similar patterns. We tried these patterns on other lists as well and had great success on the Shiro2 list, where the patterns were present as well.

```
$argon2id$x$f5MG+j5CVLwmkvmcb8kh/w$x:Maras1Maras
$argon2id$x$mgU+6EyH5mCX8psQCsSUlg$x:L'Kor1L'Kor
$argon2id$x$0lFghDp112ZXHLy1MR57Cg$x:Moklor1Moklor
$argon2id$x$46yzyphSWxHHRelq5iso8A$x:Navigator1Navigator
$argon2id$x$ApSV9FysnjYzql+nZw/EQg$x:Motura1Motura
$argon2id$x$Ga1I3Nd/p8SUeHqn2Wk3rA$x:Norman1Norman
$argon2id$x$V4q/IqnBHyXg1lAywa+UCg$x:Orgoth1Orgoth
$argon2id$x$fblze51nbi7wCGsSmtQZcA$x:Nathan1Nathan
$argon2id$x$Edjw+KGyd852rvtFJMeVtA$x:P'Chan1P'Chan
$argon2id$x$LquFJvt9zBNiI0j3hVbvaA$x:Protector1Protector
$argon2id$x$cnV7rw+Tu0CwG/ze2tj40A$x:Parent1Parent
$argon2id$x$hmBgz1+TwjDlHj+dbZFI3Q$x:Patterson1Patterson
$argon2id$x$iZviUFk71yLE4tqxxucAow$x:Q-Judge1Q-Judge
$argon2id$x$jhBgEf9KodUA46GlZPvlsw$x:Patron1Patron
$argon2id$x$p2uW80vlNU2iqKGbEIxL8A$x:Plasus1Plasus
$argon2id$x$rRhIEbG33TYicvm0TFWFyQ$x:Painter1Painter
```

3.2.10 bcr256 / bkr256

Bcr256 and bkr256 are both the same algorithm, where bkr256 start with $2k$ instead of $2b$, but is virtually identical in other aspects. Although we did try using Sha256 or Keccak-256, we did not attempt to use the salt of bcrypt as key for a SHA2-256 HMAC. The finds we were able to recover for these two algorithms were based on association attacks with usernames or hints (CRC8 from the log). This allowed us to get a small amount of cracks without the ability to properly validate the cracks. With the knowledge in hand that we had several valid hash:plain combinations we intensified our attempts to uncover what algorithm was being used, but unfortunately to no avail. Below is a code snippet of a valid find, and how it was created for future reference / those curious.

```
import hmac
import base64
import bcrypt
import hashlib
plain = hmac.new("iyvX.kDndyq/YiUvv7J.Ne".encode(), "codebreaker".encode(),
                                          hashlib.sha256).digest()
plain = base64.b64encode(plain)
target_hash = b"$2b$12$iyvX.kDndyq/YiUvv7J.NePApguNMJG65lr2k7H0A7Y3d7LLc1tOS"
print(target_hash)
print(plain)
if bcrypt.hashpw(plain, target_hash) == target_hash:
  print("Successfully Validated")
```

Chapter 4

Wishlist

Dear Santa, today was a good day because we won the contest! But there were some things that I wished could be done differently. If we're good this year, could we please have some of the following things? I know some might not be feasible or possible, but that doesn't change that they would be awesome to have! And if I'm supposed to do it myself, please have someone do it for me (or give me motivation to do my own work :)).

- Hashcat brain features 2 set to not require -S as it only transfers attack information and no plains
- A propeller cap, we'd look so cool
- An analysis tool that visually maps, and clusters similar plains together and has output options per 'cluster'
- A (wildcard) supported search tool that has libgen indexed
- Bandwidth > 10Mbps please @MyISP
- For KoreLogic to stop using PGP and maybe a more granular way to find out what hashes were invalid :)
- A team-shared archive of books, movie scripts, transcripts, and other sources
- Improved AMD support for Hashcat (HIP RTC)
- Brain client support for -a9 and –stdin
- Improved statistics automatically generated in HashMob for better analysis
- Hashcat support for 3 dictionaries
- Hashcat support for salts + peppers ex. md5($pepper.$plain.$salt)*
- Hashcat not using : as separator or change the format of 11900-12100 and NetNTLM :)
- Rule to replace the nth instance of a character (Hashcat issue 3812)
- Rule to insert a character before/after the nth instance of a character
- Rule to remove all characters of a specific charset (Hashcat issue 4030)

Chapter 5

Closing Notes

This concludes the write-up of the HashMob.net Pro team. It was a challenge at times to get everyone set up and working on the same software tools ("It works on my machine"). We have had many learning opportunities and although we were able to achieve first place, we see a lot of room for improvement in our coordination and collaboration as a relatively fresh team, with this year adding 10+ new members. We again wish to thank all members of team HashMob and the contest staff for their contributions and support. Thank you for not only taking the time to read the writeup, but also the closing notes. Finally, I'd like to invite everyone to check out the `https://hashmob.net/` website and consider joining our community, spreading the word, and joining 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. Almost all relevant questions can be answered expertly. Finally Finally (for real) we invite you to visit the first page again and take a gander in the top right corner. Enjoy, and until next year!

-Team HashMob

Chapter 6

Unfiltered Opinions

Included below are several thoughts and opinions on the contest from the team.

- **gatete:** I enjoyed this year's challenges and appreciated KoreLogic's creative ideas and patterns used to generate the hashes. Looking forward to next year!
- **mostwanted002:** Great stuff. Happy with my first time participation as a Pro Team member. Learned a lot, and hyped up for next one!
- **_0.0.0.0_:** Thank you to KoreLogic for putting on a great contest that unites the community. I had a blast and looking forward to future events.
- **stumblebot:** I especially like the idea of including real-world hash modes that are not already supported by popular cracking tools, forcing us to come up with solutions that can be integrated and used more widely. While we had some great success with them, in retrospect I'm not a huge fan of simply pulling data from online sources and using them as passwords/phrases directly. This is not reflective of how users tend to select their passwords and this felt more like a 'CTF' challenge rather than a real test of our skills.
- **Flagg:** I witnessed a lot of different challenges that engaged SMEs from many different areas, which was really cool. Sure I like the "pure cracking" part of it, but clearly we had numerous people who enjoyed reverse engineering, decrypting files, correlating cryptic clues, and developing custom crackers. So overall this competition had something for everyone. I would say it was very rough on smaller teams.
- **Cake:** An interesting mix of hashes this year. One might say it's closer to a CTF than anything. However, I feel that you might actually find this as an IRL scenario. I think that makes it more exciting to have a go at those harder hashes!
- **Vavaldi:** Glad I was able to participate this year and found the contest hashes to be challenging and fun. Overall, a job well done!
- **cyclone:** KoreLogic once again brought us a complex and engaging CMIYC contest, and I greatly appreciate the time and thoughtfulness that go into organizing these each year. The custom algorithms kept us on our toes as we wrote crackers for them, and the mix of hash cracking and cryptography was both fun and challenging (and sometimes downright annoying! Ha). However, I would prefer fewer "Alice in Wonderland" hallucinations in the contest playbook and origins of mutated plaintexts, in favor of more realistic scenarios. Based on the scoreboard alone, this year's contest left all but a few Pro teams struggling to put points on the board. As always, I'm looking forward to next year!
- **penguinkeeper:** Great CMIYC year and I couldn't be prouder of Team Hashmob,

even if it was one of the most chaotic weekends I've had in a while. Thanks to KL for another great competition! Looking forward to CMIYC 2025