CRACK ME IF YOU CAN 2022

TEAM WRITE-UP

Table of Contents

Team Hashcat

Please direct any questions or comments to hashcat Discord:

https://hashcat.net/discord

Revision 0.5
Date: 30/08/2022

Team Members

The following team members (20) were actively involved in CMIYC 2022:

atom	blandyuk	Chick3nman	dropdead	EvilMog
Hydraze	kontrast23	kryczek	matrix	m3g9tr0n
N|GHT5	philsmd	rurapenthe	The_Mechanic	T0XIC
TychoTithonus	unix-ninja	Xanadrel	xmisery	_NSAKEY

Background

Crack Me If You Can is a password-cracking contest held by KoreLogic every year at DEF CON in Las Vegas (and in uncommon cases, at DerbyCon instead of DEF CON). The contest involves various tasks related to cracking passwords that have been obtained from many different (artificial) sources. These passwords are hashed using industry-standard algorithms of various difficulty and may or may not include salts.

Team Hashcat competed in this year's CMIYC under its common name of Team Hashcat.

Team Information & Planning

The competition ran from 11 AM Las Vegas Time 12th August 2022 until 11 AM Las Vegas Time 14th August 2022. The competition was sponsored by KoreLogic as per previous years. Team Hashcat, being distributed in various geographic regions, used our consolidated platform for managing hashes, completed jobs and team communication.

Certain team members were on-site in LAS, while others operated from their respective locations. Unlike previous years, we decided to switch communication primarily to Discord.

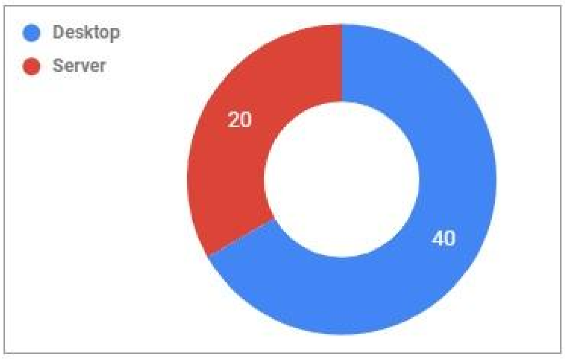
Software Stack

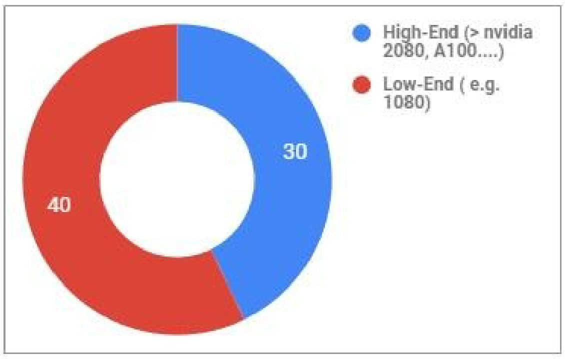
The following software was used in the competition by the Team:

Name	Version	Link
hashcat	6.2.5+664+LC	https://hashcat.net/hashcat/ (internal version that works with our collaboration platform LC)
hashcat-utils	1.9	https://github.com/hashcat/hashcat-utils
John the Ripper	latest GitHub	https://github.com/openwall/john
MDXfind	1.112	https://hashes.org/mdxfind.php
princeprocessor	0.22	https://github.com/hashcat/princeprocessor
maskprocessor	0.73	https://github.com/hashcat/maskprocessor
PACK	Python3 GitHub	https://github.com/Hydraze/pack
rling suite	1.74	https://github.com/Cynosureprime/rling
PACK2	0.1.0	https://github.com/hops/pack2
LC	-	
Google Sheets	-	

Hardware Stack

Keep in mind that in general all of our team members used just their normal hardware since this particular contest made it really hard to keep the hardware actually busy. So don't get discouraged by this quite modest list - **everybody can do it.**
The following hardware was used in the competition by the Team:





We want to again point out that the hardware used boils down to around 2-3 CPUs and GPUs per person - we deliberately do not tap into all the resources that would have been available to us unless necessary, for two reasons: 1. The challenge at CMIYC is about using brain power - 2. actually utilizing all of the resources in a smart way during a complex contest is very hard.

Updates to LC

This year our collaboration platform "LC" got significant upgrades which gave us an enormous productivity boost during the contest itself. Those upgrade include:

- automatic push of founds to LC
- push of metadata like what kind of attack has been run
- push of debug rules
- automatic retrieval of founds from LC (basically an automatic sync of all the potfile of all clients)

This was made possible by a special version of hashcat which was only available to the team. There are more advanced analytics features planned in the future but we will talk about those when we get there.

The Competition

The start of the competition

After opening up the first main container and discovering it contains more containers…

Container

PRO	
halfmd5.zip	password
20DollarDumps.odt (-m 18400)	homecoming2022
DEFCON-with-key.kdbx	keyfile only (base64 -> XML)
list23-Authorities[...].hashes.gpg	LasV3gas
LoopAESLoopAESLoopAES (unknown format, 650K)	PasswordPasswordPassword
ManMadeSelf.pdf	2022Defcon
riddle_wrapped_up_in_an.zip	Enigma22!
web.conf.tgz list5-6023_384s_-0105435TennisShoes.hash	(url found)
wopr.7z	joshua
fooo (unknown format, 3.9M random, **repeated 27 times**)	n/a [was only a filler file!]
DEFCON.kdbx (from third dl) list2-mssql05-itrustyou.hashes	Summer22
new_and_unbroken.rar	password
pennysuncle	Gadget

STREET	
list16-FL_kdIZUGpI.zip	Hackers
list17-TOWMINTP.hashes.gpg	DEFCON
list18-Thursday17January2021.odt	Sunday
list19-paidanextra500000.zip	Swordfish
list20-Authoritiesappeartohaveuncoveredavastnefariousconspiracy.7z	Queen
list24-ThisYearsWorst.pdf	Worst
DEFCON-Street.kdbx	?

halfmd5.zip - subterranean / banned

half-MD5 (5100)

The container password was guessed by hand early. This was clearly a challenge designed to be easy enough to get teams used to the idea of what was happening.

Source lists were in two groups - “subterranean” (common base words with some basic rules applied, stacked) and “banned” (entirely lower case wordlists)

20DollarDumps.odt - Kingpin

vBulletin (1611)

It took us quite some time to crack this container.

Source list was from the hacking history book “Kingpin”, for which “twenty dollar dumps” was a chapter name. This was not as easily searched for on the Internet, but we did uncover it.

DEFCON-with-key.kdbx

This container cost us much - in time and nerves. At the end of the first day, our efforts to open it were completely in vain - trying a variety of theories to open it, ranging from using parts of other containers in creative ways, did not result in anything useful. Unbeknownst to us, the keyfile wasn’t part of the initial set, but was dropped by KL during a later stage in the contest, with a hint.

The problem here is that we spend a lot of time (more than 24 hours utilizing and a little over 10 dedicated GPUs, lots of our attention and brain power) using the foofoo file as keyfile, because this file was the only file that was part of the KL work package and which was not used in any other way. The .kdbx clearly stated that it requires a keyfile so we were very sure that this is the keyfile.

The keyfile included the substring “# changed from 70617373776f7264” where the 70617373776f7264 is just the hex representation of “password”. First we thought that this is a hint for the password that typically is used in combination with a keyfile so we tried many different versions of “password” as password. Another problem was that the XML standard doesn’t allow comments using the # - so what actually happened is that the XML was rendered defective. If a clean XML parser would be used to extract the required sha256 later used as salt for the password from that XML, it would have failed. This approach was therefore very risky and also very unreal, because in a real world situation this would not occur. It is unclear to us what KL wanted to achieve with this.

However, cracking a KDBX with a keyfile only (that is, without a password in combination) failed with both hashcat and JtR. The reason here is that KeePass selects a different algorithm to calculate the KDF. Thanks to KL we have adopted this new algorithm to hashcat mode 29700 which was introduced with this commit. There is also a new tool added to hashcat called tools/recursivefiles2sha256sum.pl that recursively scans a given folder and creates a wordlist of sha 256 values of all the files inside and which then can be used to brute force an unknown keyfile.

list23-Authoritiesappeartohaveuncoveredavastnefariousconspiracy.hashes.gpg
SHA2-224 (1300)

The container's filename was a hint towards the opening sentence of a Gizmodo story about a 'hacker' homecoming queen (and so in sync with the DEFCON theme).
We wasted a bit of time on this one though, since we got a false-positive crack with "kingpinK1ngp1nKINGPIN" - thanks KL for verifying that this was in fact a collision.

The wordlist/source - only two words, 'hacker' and 'homecoming' - were simple, but the method to generate the target hashes was definitely not - or at least, the results were not easy to crack.

(We figured that it has to be continuously generating combinations of insert, overwrite and delete rules and feeding the plains and found rules back into itself but the limited time didn't allow for more exploration. - After the contest we managed to create an attack that would crack a big portion of the list just starting with the two original plains.)

Also this is where the actual competition went down once all containers have been opened - you just had to go about solving this one intelligently and not by just throwing a lot of GPU power and money at the problem. This was a really cool throwback to an older contest. We gotta hand it to you KL, good job!

LoopAESLoopAESLoopAES

SHA2-256 (1400)

For this we prepared an arch linux VM with the loop-aes packages to discover that any password would “successfully” mount the image. To find the correct one we initially tried to look for a partition table after a mount but switched to looking at the entropy with “binwalk -E”. After the switch to the entropy-method it didn’t take long to discover the actual password “PasswordPasswordPassword”.

ManMadeSelf.pdf

MySQL CRAM SHA1 (11200)

It was determined later that the title was actually supposed to be a hint for one of the other puzzles.

The single salt shared across all hashes (33313333376c33337433313333376c3333746430) is hex for the ASCII **31337**l33t**31337**l33t**d0**, which we thought might be a hint.

riddle_wrapped_up_in_an.zip / Imitation Game

SHA2-512 (1700)

The container’s name derives from a Winston Churchill quote. Once we randomly did a few cracks we discovered that the plains are based on the movie The Imitation Game, set in World War II and therefore broadly in the theme of the container quote. The screenplay was processed like so:

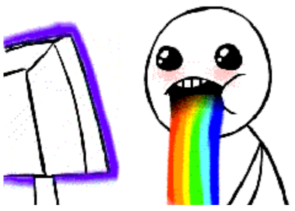
```
list6_riddle:

get imitation game transcript
extract all single words in order > imitation_game.txt_1
remove symbols > sed -e 's/[#$%*@\.\?\!:]$//' imitation_game.txt_1 > imitation_game.txt_1_1

iterate over all the words, output every word plus 4 previous words + in reverse:
while read i; do printf '%s\n' "$i"; printf '%s %s\n' "$b $i"; printf '%s %s %s\n' "$a $b $i"; printf '%s %s %s %s\n' "$c $a $b $i"; printf '%s %s %s %s %s\n' "$d $c $a $b $i"; printf '%s %s\n' "$b $i"; printf '%s %s %s\n' "$i $b $a"; printf '%s %s %s %s\n' "$i $b $a $c"; printf '%s %s %s %s %s\n' "$i $b $a $c $d"; d=$c; c=$a; a=$b; b=$i; done < imitation_game.txt_1 > imitation_game.txt_4

replace space with symbols:
hashcat.exe --potfile-disable -a0 --stdout imitation_game.txt_5 -r replace_space.rule > imitation_game.txt_5_replaced
```

```
replace_space.rule:
@
s
s !
s "
s #
s $
s %
s &
s '
s (
s )
s *
s +
s ,
s -
s .
s /
s :
s ;
s <
s =
s >
s ?
s @
s [
s \
s ]
s ^
s _
s `
s {
s |
s }
s ~
```

Team Hashcat
Please direct any questions or comments to hashcat Discord:
https://hashcat.net/discord

web.conf.tgz

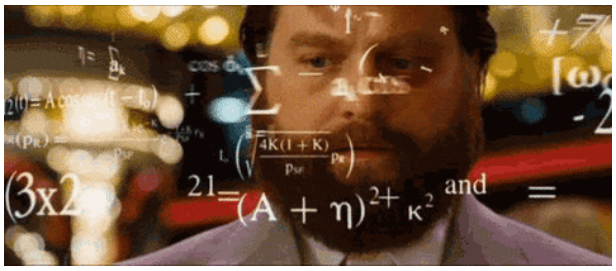
The `runme.sh` script in the archive contained a file with the interesting filename `contest_plaintexts.dic`. Our first thought was that this is kind of the golden package that if we crack this one, we can open all the (at that time) unopened other container files. We thought the encryption framework in this archive (jasypt) is of particular interest for KL and they want us to add a new plugin to hashcat. So we started looking into the output file caused by the runme.sh script which is the `output.txt` file and it contained lines like this:



pJ4Jv4eYI1KLQQgYNlhMUw==
A3EDzAVP1fJtjye1icCEJH5t7YMJmwL4
pAbFrZMxTMq4Dkx6FpALzw==

With a bit of an experienced eye you could see that this encryption is using some sort of padding, so we are dealing with an encryption/decryption system, not a hash. That made sense, since the input file is `contest_plaintexts.dic`. A closer look to the `runme.sh` script revealed the main issue here, and that is that KL (intentionally) used the password for both the encryption key but also the encrypted data in this section:

input=${id} password=${id}



The main difference between hashing and encryption when it comes to cracking is that for the encryption you have two unknown factors instead of one. You need to know either the key or the plaintext data in order to make a "test" if the key (or data) is the "correct" one (with exception to stuff like GCM). Since we know that the passwords are also the plaintext, we can assume that only two of them have a length of 8 or more. That also means that all the remaining ones had to be of length 7 or less. At that time we already had some of the other containers opened and since we assumed that all the encrypted strings in `output.txt` were encrypted passwords we assumed that one of them had to be `password`, because that was the password to the main container. That theory turned out later to be invalid, since it wasn't the passwords to all containers, but at that time we believed so. So we reduced the 12 entries down to just 2, because of the password length.

We had to step a bit into how `jasypt` works and it's really straight forward. To decrypt the content, one just uses `JasyptPBEStringDecryptionCLI` instead of `JasyptPBEStringEncryptionCLI` and that's it. We then downloaded the libraries from here: https://github.com/jasypt/jasypt/releases/download/jasypt-1.9.3/jasypt-1.9.3-dist.zip

We unpacked it in the parent folder and had to copy the lib folder from there to the current folder. We rewrote the existing `encryt.sh` to a new file `decrypt.sh` and replaced the java class to decrypting mode. Now we could run `decrypt.sh` and do our testing. We just manually tried both of them and one actually worked, exactly as we expected, because at that point we still believed to get all the container passwords from that one file.

The command line looked like this:

```
./decrypt.sh algorithm=PBEWITHMD5ANDDES input=4fk7bYelWbQlrsU1zPitfbISvKTEsluN
password=password
```

and if the password is successful, the output is something like this:

```
----OUTPUT----------------------

password
```

However, if the password is invalid, the output was like this:

```
Operation not possible (Bad input or parameters)
```

So we just built the command into a for looping shell script and started some easy passwords. For every crack we had, we stopped the shell script and replaced the hash we were looking for. It quickly turned out that some of the cracks contain pieces of a URL, for instance:

```
wWZpBNLEiXGmzgV+k5+aYQ== https
pAbFrZMxTMq4Dkx6FpALzw== .com
```

So we knew that after we cracked them all, we could form a URL and get something new from it. The final URL was:

https://contest-2022.korelogic.com/password/abcd

which redirected to:

https://contest-2022.korelogic.com/password/list5-6023_384s_-0105435TennisShoes.hash

wopr.7z -

The container password was ‘joshua’, as WOPR was the name of the computer in the movie WarGames.

inside was a list of yescrypt. We utilized JtR with ‘crypt’ format (xlibcrypt passthrough to the OS) to crack them. A couple fell pretty quickly, with the last two taking more time as people made guesses by hand or ran small lists for such a slow hash.

DEFCON.kdbx

list2-mssql05-itrustyou.hashes

The wordlist theme was a specific list of three-word inspirational phrases, with various rules applied.

new_and_unbroken.rar

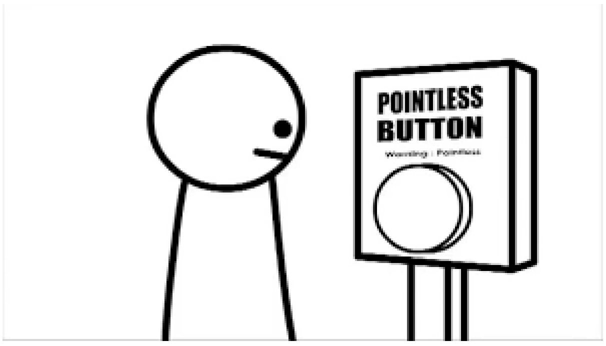
Password “passwordd”

pennysuncle

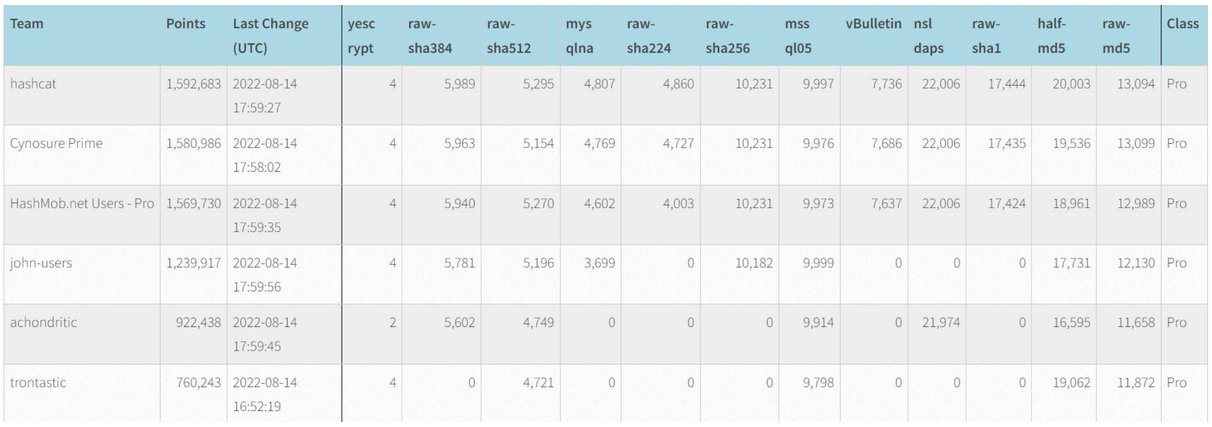
The container password - ‘Gadget’ - was derived from the hint that Inspector Gadget’s niece’s name was Penny in the 80s cartoon. The method used to discover this was basically identical to the one used to open the loopaes-container.

Street hashes

We often have some members explore these in case they prove to be useful later. We opened up the container through some manual guessing pretty quickly and cracked almost all of the hashes directly but didn’t use them since we didn’t see an obvious connection on how to apply the cracked plains.



Conclusion



Team	Points	Last Change (UTC)	yescrypt	raw-sha384	raw-sha512	mysqlna	raw-sha224	raw-sha256	mssql05	vBulletin	nsldaps	raw-sha1	half-md5	raw-md5	Class
hashcat	1,592,683	2022-08-14 17:59:27	4	5,989	5,295	4,807	4,860	10,231	9,997	7,736	22,006	17,444	20,003	13,094	Pro
Cynosure Prime	1,580,986	2022-08-14 17:58:02	4	5,963	5,154	4,769	4,727	10,231	9,976	7,686	22,006	17,435	19,536	13,099	Pro
HashMob.net Users - Pro	1,569,730	2022-08-14 17:59:35	4	5,940	5,270	4,602	4,003	10,231	9,973	7,637	22,006	17,424	18,961	12,989	Pro
john-users	1,239,917	2022-08-14 17:59:56	4	5,781	5,196	3,699	0	10,182	9,999	0	0	0	17,731	12,130	Pro
achondritic	922,438	2022-08-14 17:59:45	2	5,602	4,749	0	0	0	9,914	0	21,974	0	16,595	11,658	Pro
trontastic	760,243	2022-08-14 16:52:19	4	0	4,721	0	0	0	9,798	0	0	0	19,062	11,872	Pro

Figure 13 – Final Scores

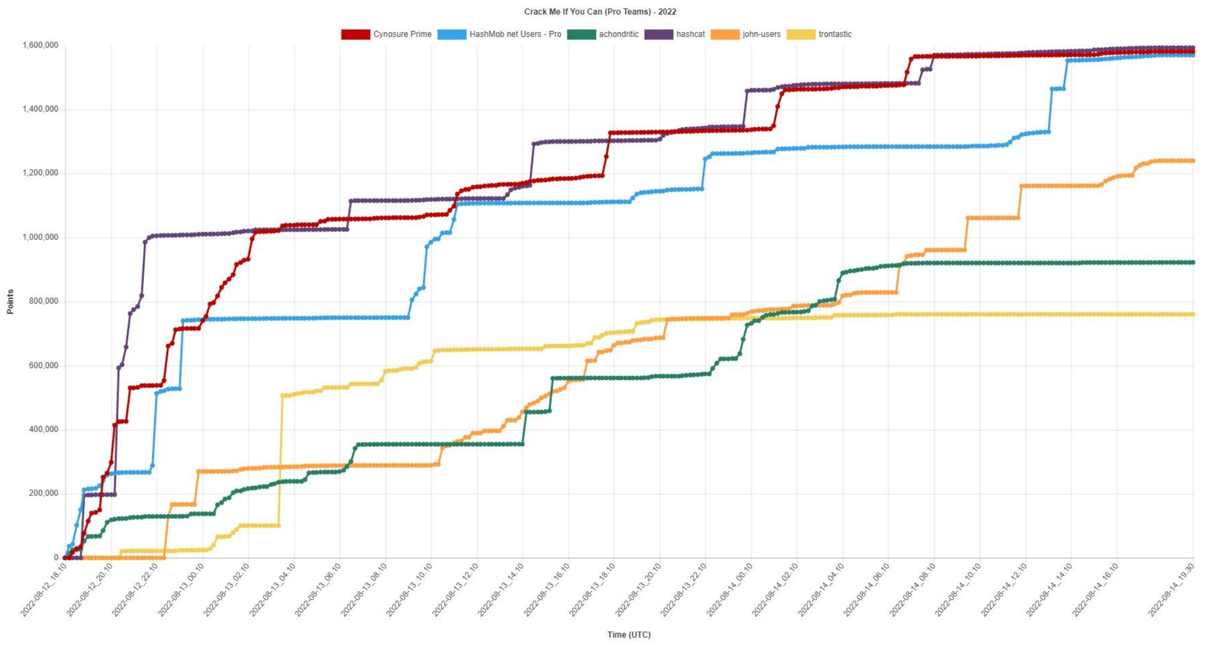


Figure 14 – Final Stats

This year's challenge was all about having creative people collaborating - it was awesome seeing and hearing the team working on the different challenges together. One moment you felt defeated and frustrated because nothing worked, while the next moment everybody on the team was excited because something new opened up - collectively, this is what this contest was all about.

Team Hashcat
Please direct any questions or comments to hashcat Discord:
https://hashcat.net/discord

Thank you KL for being an awesome host of this contest, it was a lot of “fun” for all of us. Also a thank you to all of the other teams for attending and giving us a good fight!