Crack Me If You Can 2021

Team Writeup

Revision 0.3
Date: 15/08/2021

Team Hashcat
Please direct any questions or comments to hashcat discord

Table of Contents

1 Team Members

The following team members (20) were actively involved in CMIYC 2021:

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

2 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".

3 Team Information & Planning

The competition ran from 10 AM Las Vegas Time 6th August 2021 until 6 AM Las Vegas Time 8th August 2021. 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. Contrary to previous years, we decided to switch communication primarily to Slack.

4 Software Stack

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

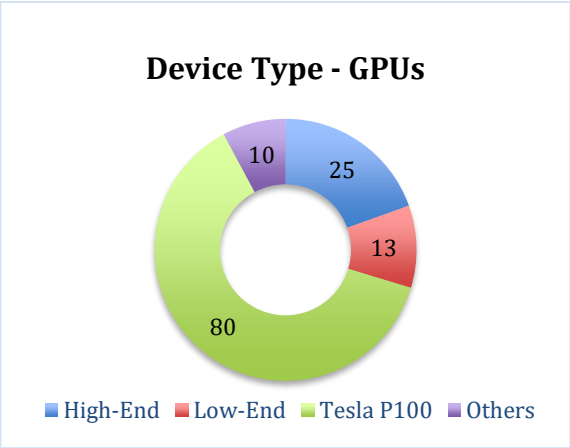
Name	Version	Link
hashcat	6.2.3	https://hashcat.net/hashcat/
Hashtopolis	0.12.0	https://github.com/s3inlc/hashtopolis
hashcat-utils	1.9	https://github.com/hashcat/hashcat-utils
MDXfind	1.112	https://hashes.org/mdxfind.php
princeprocessor	0.22	https://github.com/hashcat/princeprocessor
maskprocessor	0.73	https://github.com/hashcat/maskprocessor
hashcat-legacy	2.0.0	https://github.com/hashcat/hashcat-legacy
Impacket	0.9.23	https://github.com/SecureAuthCorp/impacket
DSInternals	4.4	https://github.com/MichaelGrafnetter/DSInternals
Mimikatz	2.2.0-20210729	https://github.com/gentilkiwi/mimikatz

qemu	1.4.2	https://github.com/qemu/qemu
autopsy	4.19.0	https://github.com/sleuthkit/autopsy
LC	-	
Google Sheets	-	
stuff self-made during contest	-	https://github.com/hashcat/team-hashcat/events/CMIYC2021/

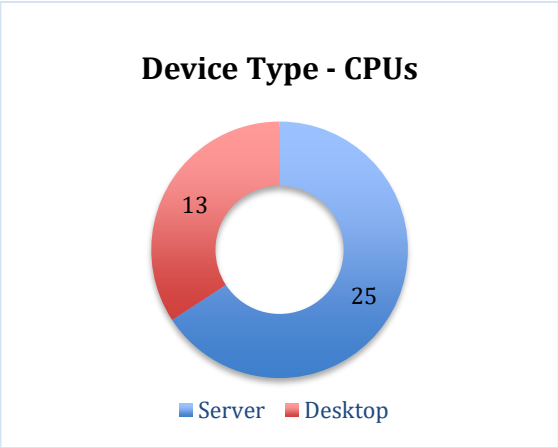
5 Hardware Stack

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:



128 GPU's



60 CPU's

6 The Competition

6.1 The start of the competition

Korelogic decided to throw a curveball this year by not providing hashlists directly, but instead providing an OVA image containing a Windows Server 2019 VM enabled as an Active Directory domain controller. To make working with the VM hard, Korelogic decided to booby trap it with various things like automatic shutdowns, calling back to KL (if networking was enabled), replacing various menu entries with garbage, etc.

From our side, extract hashes were easy; after import the OVA into VMware we mount the vmdk disk.

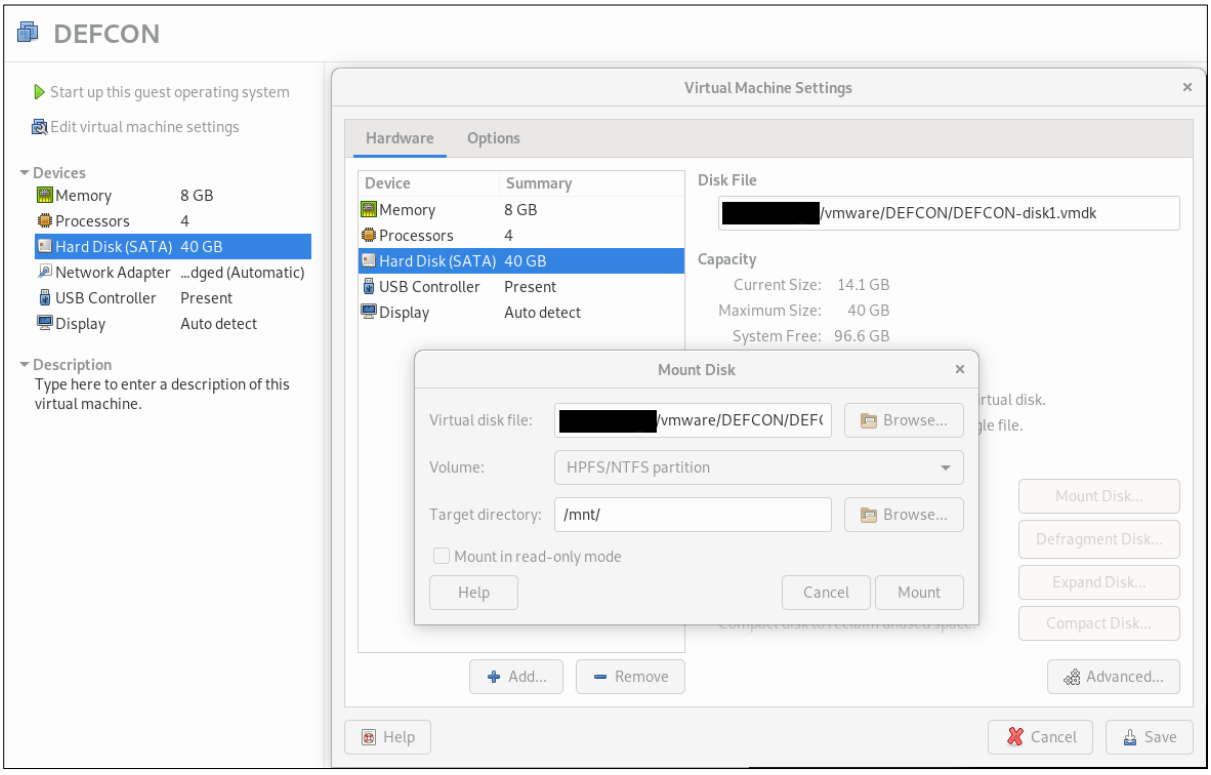


Figure 1 – mounting vmdk disk

Next we make a copy of SAM, SECURITY, SYSTEM and ntds.dit files and then we use secretsdump from Impacket to perform extractions of local and domain hashes.

```
$ cp -avf /mnt/Windows/System32/config/{SAM,SECURITY,SYSTEM} /tmp/
'/mnt/Windows/System32/config/SAM' -> '/tmp/SAM'
'/mnt/Windows/System32/config/SECURITY' -> '/tmp/SECURITY'
'/mnt/Windows/System32/config/SYSTEM' -> '/tmp/SYSTEM'
$ cp -avf /mnt/Windows/NTDS/ntds.dit /tmp/
'/mnt/Windows/NTDS/ntds.dit' -> '/tmp/ntds.dit'
```

Figure 2 – extract SAM, SECURITY, SYSTEM and ntds.dit from vmdk

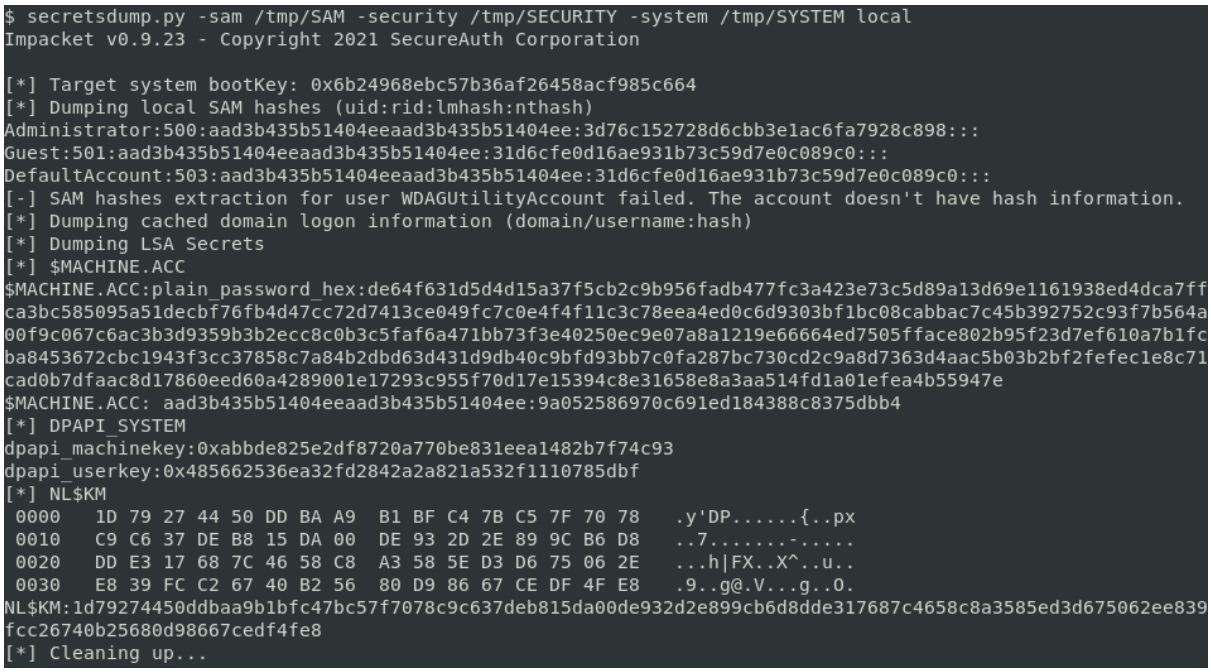


Figure 3 – extracting local hashes with Impacket/secretsdump

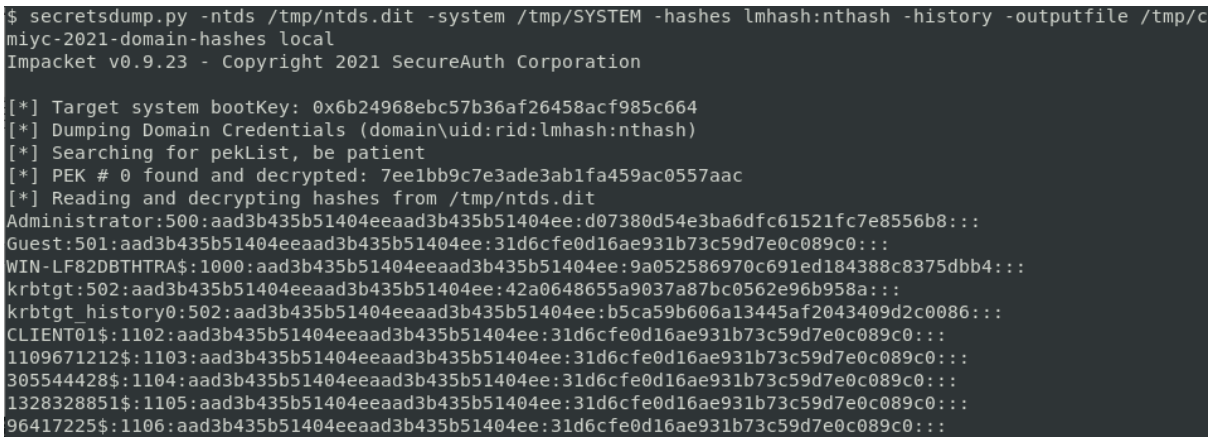


Figure 4 – extracting domain hashes with Impacket/secretsdump

During the full extraction of domain hashes, we successfully cracking the local admin password and then the domain one. With these passwords we are able to login to Windows, and found the “plaintexts” :)

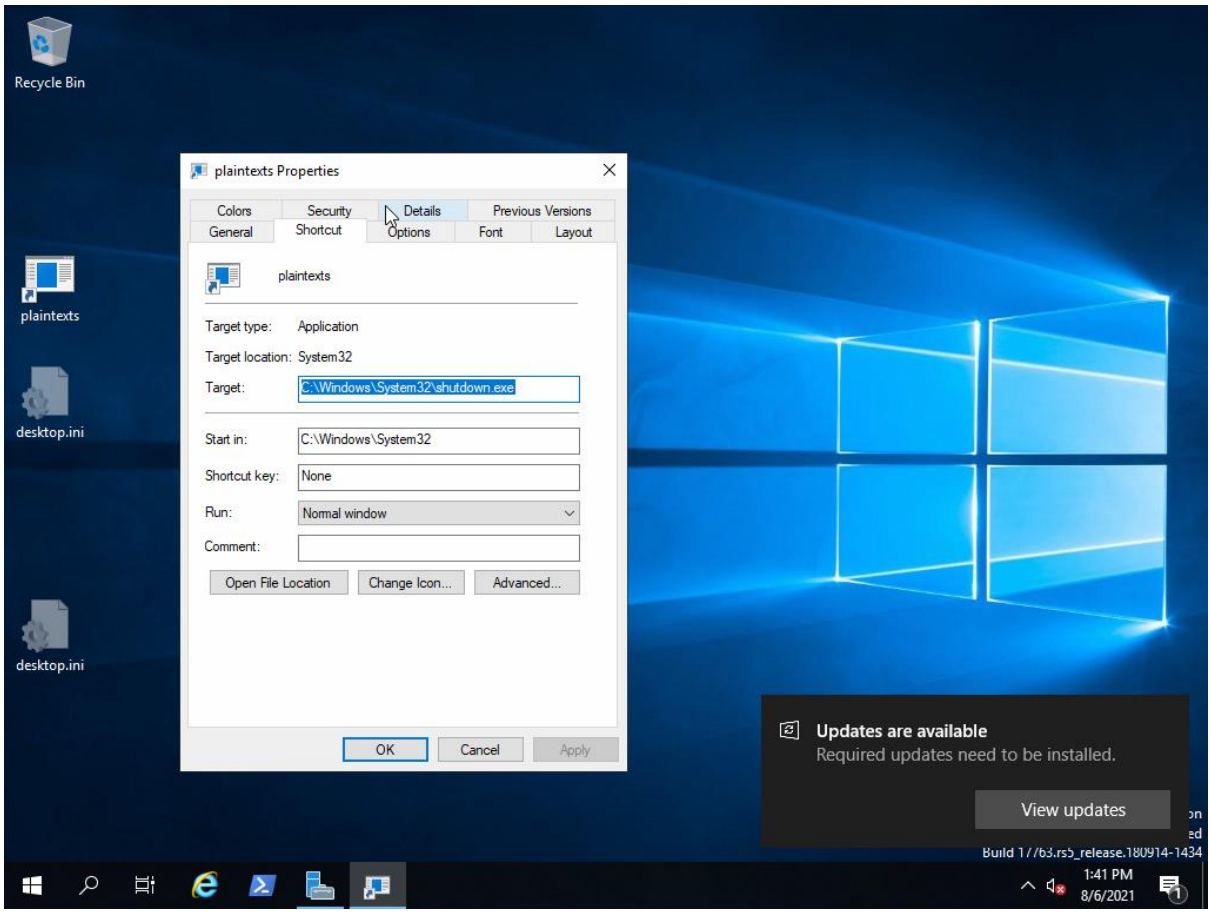


Figure 5 – plaintexts Properties

We proceed first by remove Windows Defender using Server Manager (from Manage/Remove Roles and Features) to disable Antivirus real-time protection.

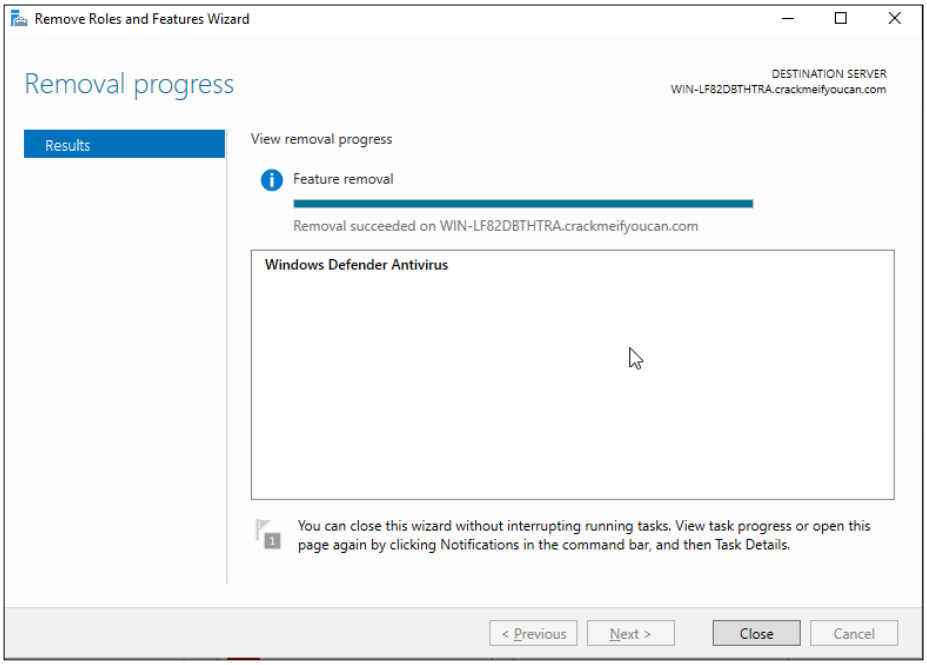


Figure 6 – removed Windows Defender Antivirus

Then we connect a USB drive to VM and we are able to run mimikatz. In that way we are able to extract some others/related info useful to perform future analysis/attacks.

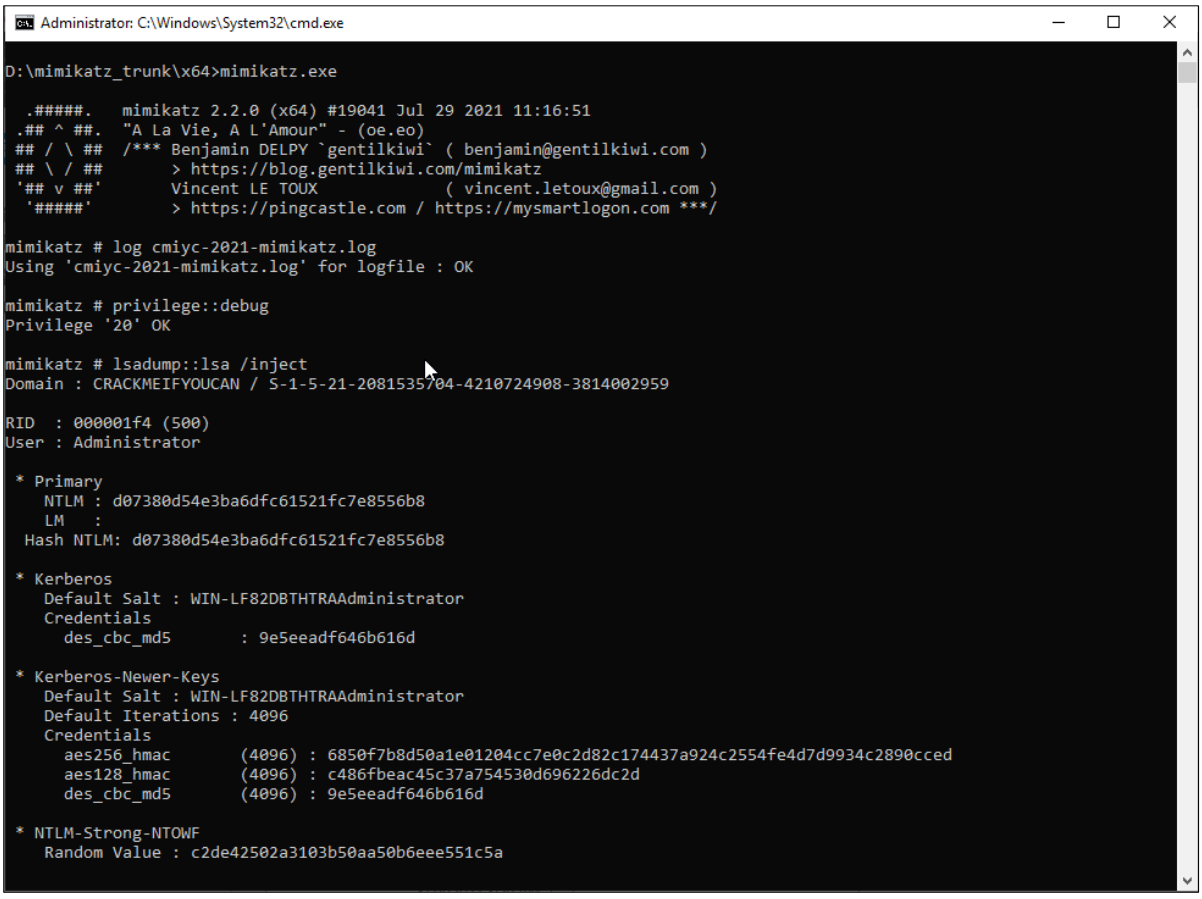


Figure 7 – extracting credentials using lsadump with mimikatz

Notes:

- to use a USB drive we had to set "USB Compatibility" to "USB 3.1" in the VM settings on VMware
- Windows server shutdown every hour due to license expired. We fix this using Powershell: “slmgr -dlv” and “slmgr -rearm”

To make sure about all hashes was extracted, we use DSInternals Get-ADDBAccount powershell script to dump again the ntds.dit, after repairing it using “esentutil”.

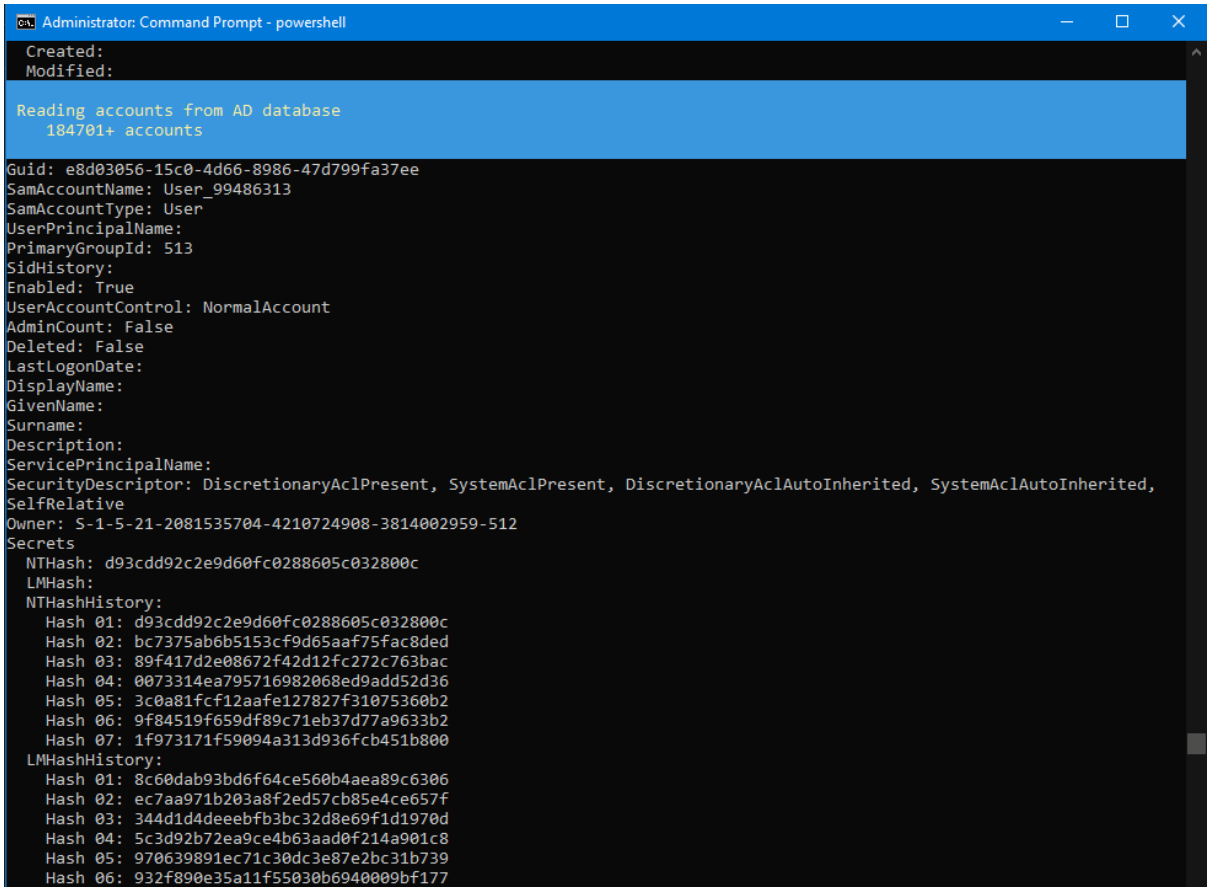


Figure 8 – extracting accounts with secret attributes from AD using DSInternals Get-ADDBAccount

We are using filter to retrieve only have the user objects (remove the computer objects). This helped to also get the "Description" field with passwords in clear text.

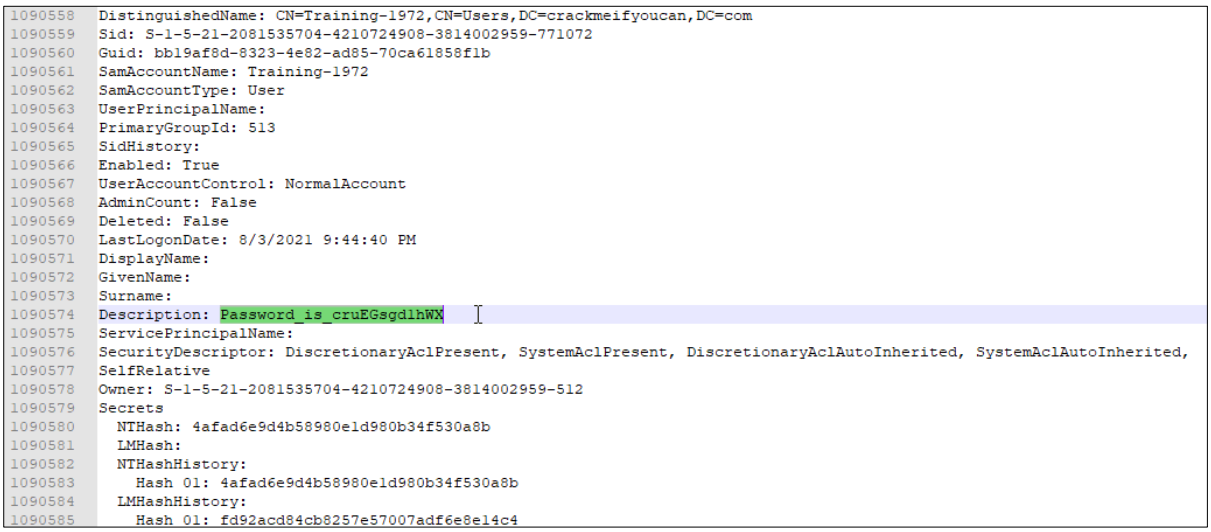


Figure 9 - extract Training's user password from DSInternals / Get-ADDBAccount dump

These passwords have been extracted also using strings from linux, as shown below.

```
$ strings -el /tmp/ntds.dit | grep -i password | tac
Password_is_HXMV0YzeKfth
Password_is_N6HRGpBjChGk
Password_is_8Cp4YHQhaGdI
Password_is_3fvqv4guq3UG
Password_is_DhD4TAXb8Gcp
Password_is_jMmLfi1dtTZG
Password_is_9gzLI7LZDRrK
Password_is_DMFeHgi46eiR
Password_is_ZEDnB4sCQ09a
```

Figure 10 – extracting Training's user password from ntds.dit with strings

6.2 Hashes

This year, the number of hashes in the competition were relatively small compared to previous years – but not any easier. The hashes were spread out as follows:

- NTLM hashes at ~70 000 hashes

However, during the whole contest we were not sure if these 70k were actually all of them due to Korelogic's mindfuckery.

All hashes came out of the active directory with username and correlated history, with "history0" being the user's current password, through "history6" as the oldest. This is also how Korelogic awarded points - value of the hash increasing with the complexity:

History Entry	Points Each
history0	64
history1	32
history2	16
history3	8
history4	4
history5	2
history6	1

This led to an interesting dynamic, since the oldest (and lowest scoring) hashes needed to be cracked first to extract the information necessary to progress to the newer ones.

Unsurprisingly, History6 list did not give us any trouble. We basically managed to crack almost all of them instantly with standard attacks like rockyou+rules. The same goes for History5.

As a side note, the initial few hours were especially chaotic due to Korelogic releasing the lists for history6 and history5 in the first couple of hours, which forced us to double-check whether our lists were correct. So good job on introducing another layer of chaos, @KL :P

At the same time, history4 was slowly being cracked - by iterating over the founds from history6 and history5, using additional rules and hybrid attacks as patterns emerged.

The first layer that began to give us more of a challenge was history3. At this point in the contest, it was clear that the same process demonstrated in the test hashes (released prior to the contest) was used on the contest hashes as well and that every "newer" layer (history 2, 1 and 0) was based on further manipulations of previous founds, either directly (additional rules applied to the preceding found) or indirectly (general thematic trends per user).

At this point, we began a Google spreadsheet tab to correlate the cracks we have against the users from where they came.

A	B	C	E	F	G	H	I	J	K	M	O
User	Type	ID	Current-found	H0-found	H1-found	H2-found	H3-found	H4-found	H5-found	Cracks	Source?
User_10008724	User	10008724		1AiR_hoCkey-e		Cosmic(1	XboxOneD1@	MAhjONG0	nphess	5	_games_
User_10021922	User	10021922	KoCTOber835)	noVeMberjun6	YNOVember574@	95-OCtober	8109AprII	mAY62371	aprilyy	7	_month_
User_10029329	User	10029329	Raymond1933,	FordAbby0486	KARnage974!!	1Bjorn3	10031992Lg	001025356	saltland	7	_names_
User_10030254	User	10030254	0BS3S$IVENes$D	0bS3S$IvepNes$	hB$E$$IVEN3ss	Bo4$njovi	BonJo9@vi	21626472	talkvnm	7	_bonjovi_
User_1003871	User	1003871	jaNUary`9734	FEbRUary=939	xjAnUary616!	aPrII3795	vmArcH830{	096aPRII5	juneee	7	_month_
User_10048146	User	10048146				Squish29!	NintendoH1	Channca37	aarrwor	4	_games_
User_10048642	User	10048642	AlejandrO7902)(	Mavericks_41	Siddhant#007	Timothy.2	Adam0402	Alana2223	jeanic	7	_names_
User_10077517	User	10077517	T53r3f0r3 wit5in is m3 53@rt	Therefore within is me heart	overwhelmed within me	my spirit overwhelmed	overwhelmed	Psalms 143:4	Psalms68	7	_bible_
User_10087739	User	10087739		SpaceInv.1ers1	Dotstrea@390	Bas0ball-3	FlingSma!8	Manholec1	qsuipa	6	_games_
User_10101651	User	10101651	SolitaireDSi#157			BOXbOY!31	_3_Mystery=4	Motara3sw	ithaasuw	5	_games_
User_10119951	User	10119951		@ElRos@rio20219	He@lthc$@re20218	Santi#ago3	Maraca$ib5	Abriaquí	ecuador9	6	_latinloc_
User_10149895	User	10149895	Fiftymillion1990#	2007Tensixty	Thirty2011-	F=vef0ur	Tenseve7	22thirtyfive	nineforty	7	_numberword_
User_10169946	User	10169946	UnOVember725(	tu3sDAySepTember	70,SEPtember	ApriL75@@	maRCh*600	4531APRII	sasyjuly	7	_month_
Training-10203	Training	10203	pgagRFIITyUo							1	_training_
User_10203026	User	10203026	januAry#4293	feBRuaryX054	NOVEmber_108	aUguSt66??	aUG1928ust	aPRiLV755	januarn	7	_month_
User_10221704	User	10221704			fastest, anoint thine	when thou fastest,	fastest	Matthew 6:17	Matthew16	5	_bible_
User_10228944	User	10228944	Theodore@2010	Sya22Michelle	mIGUELaNGEL.001	Andrea1,4	Eric2206	Jeremy495	haroldg	7	_names_

(Due to an initial misunderstanding, our internal names for the lists in our shared collaboration server were “off by one” in their naming (current == history0 on our side, history1 == H1 on our side and so on...). However, our spreadsheet ignored our internal representation, relying solely on the exact history strings embedded in the usernames.

From that initial correlation tab, we began to identify different groups (or themes) that the plains were based on. As the groups became clear and new cracks became available, we began to iteratively repopulate the spreadsheet for central reference.

Using a perl script that we wrote on the fly on Saturday, we analyzed each user on a per-crack basis and applied a heuristic to make a cumulative “best guess” as to which password group that user belonged to. For example, if three passwords for a given user contained number words like “billion” and “sixty”, then that user received a “numberword” score of 3. The largest score was then used to “vote” for the “best” category for that user.

There were some false positives due to certain substrings being used at first (with “two” being one of the more problematic, so it was eventually omitted), but the heuristic, while simple, was generally pretty reliable - and got increasingly more reliable as more cracks arrived and with some additional tuning.

The resulting labels allowed any member who wanted to attack or study a specific group to filter on just that group, reducing the noise of unrelated cracks and allowing natural pattern recognition “by sight” to be much more efficient.

The identified groups were initially tagged as follows (though some of them ended up being slightly inaccurate, they were still useful for filtering the target hashes):

bible	bonjovi	ferengi	games	song	numberword
contest	latinloc	password	h2stem	training	website

We are going to refer to these groups from here on out.

6.2.1 KJV Bible

This group identified itself by having a history6 and history5 crack to a name plus a number which turned out to be psalms of the bible. To move up a step to history4, we needed first to crack using a couple of standard methods. Once we had a couple of examples, we could correlate the psalms and identify that the plain is a word from the psalm.

So far, so good - but what about history3 and above? There it started to get a bit tricky. For a while nothing worked, but with some endurance, we managed to crack a couple of the history3 to find that those turned out to be 3 words from the psalm. Once identified, extracting those 3-word phrases from a sourced KJV bible could be done with a simple script (1250 total cracks).

From here, going to history2 was another layer of complexity which proved to be challenging at first. Finding the first cracks took quite a while, but by random chance we found a few that turned out to be similar words but different.

After further analysis and tests the revelation was that those crack to 3-5 words from the psalm, seamling random in order but we knew at this point where this was heading. At this point, we had an idea but feared that the potential keyspace (if you permuted every combination of words from every psalm for a particular user) was quite high - and labor intensive. The solution was to automate - first parts of it, then everything.

The first iteration was a perl script that converted a psalm into a hashcat-legacy table attack. By adding some additional modifiers to hashcat, it also gave us the first cracks of history0 - which turned out to be exactly the same text as history1 but with leetspeak.

its closing time -> 1+5 c!051ng +1m3

Knowing this we started another iteration of the attack.

The second iteration was to exchange hashcat-legacy with the freshly created tool “simple-table” that atom prepared before the contest (because of the test hashes) which looked like this:

```
cat rawphrase.txt | tr ' ' '\n' | sort -u |sed -e 's/^/1\t/' |sort -u > table;
cat table.add >> table;
perl simple_table.pl template table | hashcat -m 1000 all.left -r base.rule -r leetspeak.rule --username -o current.out
```

Additionally we created *base.rule* which contained overall modifiers that we found could be used - like removing the character “j” - and *leetspeak.rule* which contained the identified leetspeak-conversions.

leetspeak.rule

```
sa@s<cse3si1so0ss$sh5sv^sl!st+
sa4sc<se3si1so0ss$sh5sv^sl!st+
sa@sc<se3si1so0ss5sv^sl!st+
sa4sc<se3si1so0ss5sv^sl!st+
sh5 so0 se3 sa@
sh5 so0 se3
sh5 so0
sh5
so0 se3 sa@
se3 sa@
sa@
so0 se3
sh5 so0 sa@
sh5 se3 sa@
sh5 sa@
st+
sh5 so0 se3 sa@ st+ sv^
sh5 so0 se3 st+
sh5 so0 st+
sh5 st+
so0 se3 sa@ st+
se3 sa@ st+
sa@ st+
so0 se3 st+
sh5 so0 sa@ st+
sh5 se3l sa@ st+
```

Continuing the attack with the second iteration was working, but still not very efficient - due to the effort required in precisely identifying the psalm. And depending on the word count in the psalm, the very large keyspace that the attack needs to go through to find both plains for history1 and history0.

Further iterations:

permute3.c	Tool version 3 for applying permutation attacks like permute.c from hashcat-utils, but not for individual letters of passwords, but for words.
splitter.pl	Extract 5/N random words for a Bible psalm The speed of word generation is around 22 MW/s per thread *$ time ./permute < 100.txt | head -1000000000 > /dev/null* *real 0m45.005s* The speed drops to 1.5 MH / s due to parsing, expanding, compressing and finally hashing The total key space of all 5 word permutations from all KJV psalms (with our reference database and without uniqueness) is 1,617,781,809,660

	Then 1617781809660/1500000 = 1078521 seconds = 300 hours = too much
minintlm3.c	As a solution to this, we had to improve the I/O time so that we could achieve 22 MH/s instead of 1.5 MH/s. The result is a standalone NTLM cracker, threadless, only CPU, but very light, so it can execute N many instances. Why not use Hashcat? The splitter.pl script was too slow to generate enough candidate passwords quickly enough while running on a single instance. Hence, we need to run multiple instances of it. But multiple instances of Hashcat put a heavy load on the system. So why not use jtr/mdxfind? Both do not support NTLM with password candidates > length 27, which was very often the case with the KJV challenge. The three tools are chained as follows: *$ splitter.pl kjv_all.txt |./permute3 | ./minintlm history0_hashes.txt*
history0.c	A tool developed after we had some success with the above chains but works more efficiently. It combines permute3 and minintlm and does a unique sorting of the words per psalm to further reduce the number of combinations. It now contains essentially all the relevant code to write a simple password cracker. We've added some comments so that anyone interested in how a password cracker is built can easily learn from it. Of course, this doesn't include deep tweaks, threading, or writing crypto primitives as the goal was to keep it simple
bla2.sh	Simple script that is run N times history0 to make full use of all CPU resources. It divides the workload among the threads by using the hashcat-utils' gate.bin utility. For example, my computer has 16 CPU listed in /proc/cpu, so I use "gate.bin 16 0" for the first thread, "gate.bin 16 1" for the second, and so on. The same could be done in history0, but it's a good opportunity to show how gate.bin can help parallelize programs that don't natively support multiple threads

Tools can be found here: https://github.com/hashcat/team-hashcat/events/CMIYC2021

When things cleared up, we had a total of 605 cracks from history0/1.

If you are wondering how we got a good source to programmatically use as a basis for these attacks, we pulled it from the Gutenberg Project and processed it a bit. Due to copyright issues, we cannot do more than linking to the source: https://www.gutenberg.org/files/10/10-0.txt

You cannot use this source 1:1 due to formatting issues like word wrapping. We used regex-magic to untangle it.

6.2.2 Bonjovi

BON JOVI :D and variants on the word “Obsessiveness”

6.2.3 Ferengi

Ferengi Rules of Acquisition (Deep Space 9), reordered, then with rules. This group was fairly straight forward to crack. Once we had enough cracks, it was just a matter of doing a Google search with the exact number and format of how it's written to find the actual source.

history5	Number.	239.
history4	Rule #Number.	Rule #239.
history3	Word + ?d?s	mislabel1
history2	3-5 words from the rule in right order + ?d	afraid to mislabel1
history1	3-5 words from the rule in random order	mislabel a product.1
history0	we didn't crack anything here	afraid be a mislabel to

Note: for anybody interested, after the contest was finished, we figured out the actual replace-rules to crack history0 by running the plains from history1 against a table attack with leet-table. One by one we opened them and extracted the rules:
seE sl1 si: so. st% ssS sa@ sx sg&*

6.2.4 Games

This group went kind meh, so here are just the notes that survived the chaos:

Games - board games, video games, card games - may even be separate subgroups	https://nintendo.fandom.com/wiki/List_of_Nintendo_games and others
Apply some rules with hashcat using cpu, then pipe to gpu w/more rules	lots of "game"-related ones available. download correlation tab and filter on "game" surrounded by underscores
	strip spaces and trim at length 8

6.2.5 Song

Should have been divided between song lyrics and author quotes, but in practice this was very difficult to automate. In the debriefing, it was clear that KoreLogic generated these as a single group of hashes on purpose.

Cracking history5-2 was not that difficult, and mostly done by standard methods. History1 and 0 was again quite hard to get cracks but after the first few we saw a similar pattern appear as for the bible.

history6	name + ?d	Arnold1
history5	full name + ?d	Tim Arnold1
history4	word that relates to the name + ?d	would1
history3	2-3 words that relates to the name + ?d	what love would1
history2	2-3 words that relates to the name mixed with ?d?s	would want1
history1	3-5 random words that relate to the name	love want would what Ask
history0	3-5 random words that relate to the name + leetspeak.rule	!0v3 w4n+ w0u!d wh4+ A5k

Since we didn't manage to get a hold of a good source for either author quotes or songs to crack a lot of those, we were not able to really elaborate on the exact pattern.

6.2.6 Numberword

These were all based on the written forms of some different numbers, including "two", "billion", "sixty", and others. Proceeding from the oldest to newest was straightforward, appending numbers, extending length, and applying rules.

6.2.7 Contest

DEF CON, Vegas, Ballys, etc etc etc

6.2.8 Latinloc

These were locations in Latin America, including states in Mexico, departments in Bolivia, and others.

6.2.9 Password

Passwords regularly based on the word "password", but also a mashup of general password strategies. May be based on RockYou or some other corpus. This group is probably what the KL out-briefing referred to as the "intern-generated" hashes.

6.2.10 h2stem

h2 column has the stem word - always [word]2020 or 2021, with optional single special.

6.2.11 Training

As shown on chapter 6.1, we extracted the plaintext from ntds.dit and they match 1:1 with Training's user passwords.

6.2.12 Website

Did not crack any - this puzzle totally escaped us. The information required had to be pulled from other metadata associated with specific users.
After the end of the contest, we successfully extract the website's user passwords from ntds.dit using strings :)

```
$ strings -el /tmp/ntds.dit | grep https
https://6477:3C0Zu8GkMPTVLukL@www.crackmeifyoucan.com/
https://3956:o2xSjyFAB2lJCtYe@www.crackmeifyoucan.com/
https://5272:s3fGhLQTr1LFR3bV@www.crackmeifyoucan.com/
https://3438:ty7RfMQEI4MWMPtr@www.crackmeifyoucan.com/
https://20415:ZDhFzJw4CsbE35IS@www.crackmeifyoucan.com/
https://12981:ActVsIZudw83Zo8L@www.crackmeifyoucan.com/
https://18475:8jmD3o612mDRdaTy@www.crackmeifyoucan.com/
https://25033:2kA7PyyjzK2eLaVA@www.crackmeifyoucan.com/
```

Figure 11 - extracting Website's user passwords from ntds.dit using strings

111 plaintexts useful to crack 1:1 history0 hashes, so 7104 missing points here :|

6.3 Other stuff we did

6.3.1 Autocrack for sleep time:

autocrack.sh

```
#!/bin/sh
while true; do
  shuf -n 10 dict_all.dict.txt | smartquote.pl| sort -u |sed -e 's/^/1\t/' > table
  shuf -n 2 template.seed > template
  perl simple_table.pl template table | hashcat/hashcat -m 1000 all.left -g 100000 -o
current.out --remove --generate-rules-func-max=30
  hashcat/hashcat -m 1000 all.left dict_all.dict.txt -g 10000000 -O -w3 -o current.out --
remove --generate-rules-func-max=30
done
```

template.seed

```
11
111
1111
11111
1 1
1 1 1
1 1 1 1
1 1 1 1 1
```

6.3.2 “Blind” non pattern attacks based on all cracks

blind.sh

```
#!/bin/sh
cat dict.txt > 1.txt
sort dict.txt | tr -d [:punct:] | tr -d [:digit:] | uniq -ic | sort -rn | cut -b9- >> 1.txt
sort dict.txt | tr -d [:punct:] | tr -d [:digit:] | tr [:upper:] [:lower:] | uniq -ic | sort -rn | cut -
b9- >> 1.txt
cat dict.txt | tr [:alpha:] "\n" >> 1.txt
cat dict.txt | tr [:digit:] "\n" >> 1.txt
cat dict.txt | tr [:punct:] "\n" >> 1.txt
cat dict.txt | tr [:punct:] "\n" | tr [:digit:] "\n" >> 1.txt
cat dict.txt | tr [:punct:] "\n" | tr [:digit:] "\n" >> 1.txt
cat dict.txt | tr [:upper:] "\n" >> 1.txt
cat dict.txt | tr [:lower:] "\n" >> 1.txt
cat test.wordlist >> 1.txt
```

```
cat anothertest.wordlist >> 1.txt
cat 1.txt | sort | uniq -ic | sort -rn | cut -b9- > 2.txt
cat 2.txt > 3.txt
cat 2.txt | cut -b -3 >> 3.txt
cat 2.txt | cut -b -4 >> 3.txt
cat 2.txt | cut -b -5 >> 3.txt
cat 2.txt | cut -b -6 >> 3.txt
cat 2.txt | cut -b -7 >> 3.txt
cat 2.txt | cut -b -8 >> 3.txt
cat 2.txt | cut -b -9 >> 3.txt
cat 2.txt | cut -b -10 >> 3.txt
cat 2.txt | cut -b -11 >> 3.txt
cat 2.txt | cut -b -12 >> 3.txt
cat 2.txt | cut -b -13 >> 3.txt
cat 2.txt | cut -b -14 >> 3.txt
cat 2.txt | cut -b -15 >> 3.txt
cat 2.txt | cut -b -16 >> 3.txt
cat 2.txt | cut -b -17 >> 3.txt
cat 2.txt | cut -b -18 >> 3.txt
cat 2.txt | cut -b -19 >> 3.txt
cat 2.txt | cut -b -20 >> 3.txt
cat 2.txt | cut -b -21 >> 3.txt
cat 2.txt | cut -b -22 >> 3.txt
cat 2.txt | cut -b -23 >> 3.txt
cat 2.txt | cut -b -24 >> 3.txt
cat 2.txt | cut -b -25 >> 3.txt
cat 2.txt | cut -b -26 >> 3.txt
cat 2.txt | cut -b -27 >> 3.txt
cat 2.txt | cut -b -28 >> 3.txt
cat 2.txt | cut -b -29 >> 3.txt
cat 2.txt | cut -b -30 >> 3.txt
sort 3.txt | uniq -ic | sort -rn | cut -b9- > 4.txt
cat 4.txt > 5.txt
cat 4.txt | tr [:upper:] [:lower:] >> 5.txt
sort 5.txt | uniq -ic | sort -rn | cut -b9- > 6.txt
cat 1p.txt >> 6.txt
sort 6.txt | uniq -ic | sort -rn | cut -b9- > wordlist.txt
```

Then run:

- -a 6 -a 7 attacks with mask like ?a?a?a?a
- -a 1 attacks | stdin with debugged rules from previous cracks
- combined rules attacks -r toggle -r debugged
- -a 1 attacks with phrases (2,3,4,5 words) and -j $[space]

7 Conclusions

Team	Points	Last Change (UTC)	history0	history1	history2	history3	history4	history5	history6	Class
Hashcat	978,070	2021-08-08 16:58:51	7,497	7,538	7,913	8,687	8,722	8,710	8,634	Pro
Cynosure Prime	949,848	2021-08-08 16:57:07	6,963	7,691	8,060	8,598	8,606	8,654	8,628	Pro
HashMob.net users	831,122	2021-08-08 16:59:16	5,860	6,506	7,436	8,539	8,663	8,667	8,616	Pro
john-users	800,430	2021-08-08 16:59:50	5,849	5,926	6,882	8,400	8,332	8,600	8,622	Pro
trontastic	778,391	2021-08-08 16:59:02	5,405	6,081	6,976	8,313	8,571	8,492	8,491	Pro
1IHsxRAM7GzoM	630,694	2021-08-08 16:49:37	4,101	4,878	5,756	7,802	8,052	8,414	8,586	Pro

Figure 12 – Final Scores

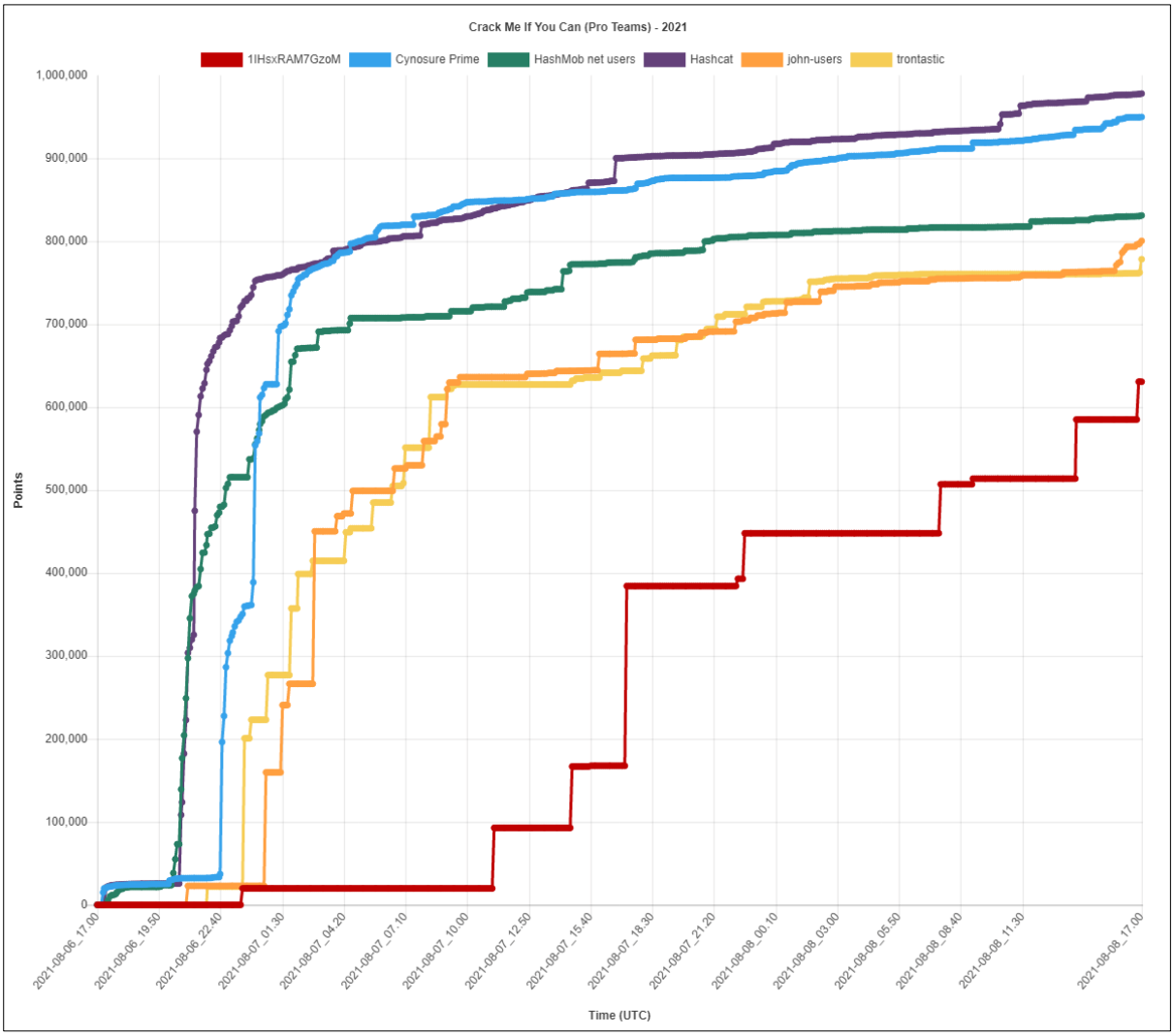


Figure 13 – Final Stats

1/10, would not bang… Besides that, pretty cool 48 hours, thanks KL!

Here are some unfiltered thoughts of our team members:

evilmog: The contest was chaotic, while I liked the fact that it was only NTLM it was extremely hard to get into my groove. Most likely I'm out of practice, but my frustrations for the most part are to do with me fighting with LC and us having no clean way of exploiting our massive pile of hardware. Next year I'm just not using htp and will have to code something custom to send manual attacks out. It seems every year Minga screws with the pro teams and it's kind of lots of fun. The Pro contest take so much time to get into the groove that you miss out on the rest of defcon.

matrix: I enjoyed it a lot, especially because of the VM challenge, for which some skills related to the world of security were needed.

Xanadrel: Spent first hours monitoring & fixing shit, didn't even play with the VM, didn't do much cracking, this felt like the dullest contest I ever did.

_NSAKEY: The AD stuff turned me off, but I get why it was done, and a Category 4 Doom Storm at work pulled me away during the last 3 hours.

Kontrast23: interesting contest, partly over-engineered, constant impression "there must be more" but wasn't, enjoyed the hunt and development

unix-ninja: there was a lot of dissonance; communication wasn't great and there as a bit of overlapping work.

atom: The KJV challenge has been my personal favorite challenge. I had to write several small tools to get there, but for the short time it was, it wasn't too difficult. And the reward was good enough to spend most of the time on. Still, I wonder if I could do better if I took advantage of the RNG involved. But without the possibility to analyze a source or binary it is very likely I am just wasting my time.

tychotithonus: I enjoyed the novelty of working with purely fast hashes, the variety of strategies used to generate the target plaintexts, and the challenge of correlation of user password history (which I didn't have much experience with).

blandyuk: Good contest but would have liked more algos like MSSQL and maybe some phpass on a Wordpress site. This could have easily been done via a second VM. The patterns per user which got harder as the history went down was great. More complex but still based on original password. Shows users do not go far from password.

RuraPenthe: I liked the real-world scenario of a damaged or lost domain controller image we had to recover to get the hayes that Minga mixed with his usual evil ways to make us suffer.

dropdead: Loved the balance between "too hard" and "super easy" in combination with sheer willpower to break through those histories.