CRACK ME IF YOU CAN 2021

TEAM WRITE-UP

Table of Contents

Team Hashcat
Please direct any questions or comments to hashcat discord

Revision 0.3
Date: 14/08/2021

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

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 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.

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/secretsdump	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
stuff made by us during the contest	-	https://github.com/hashcat/team-hashcat/events/CMIYC2021/
qemu	1.4.2	https://github.com/qemu/qemu
vbox	6.1	https://www.virtualbox.org/
autopsy	4.19.0	https://github.com/sleuthkit/autopsy
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:

DEVICE TYPE - GPUs	COUNT
High-end (>= NVIDIA 2080, A100, etc)	25
"older" GPUs (e.g. 1080, 980ti, titan, etc)	13
NVIDIA Tesla P100 (and Others)	90
TOTAL GPUs	128
DEVICE TYPE - CPUs	COUNT
Server grade CPUs (e.g. Epyc 7000, Intel Xeon Cascade Lake)	20
Desktop grade CPUs	40
TOTAL CPUs	60

The Competition

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.

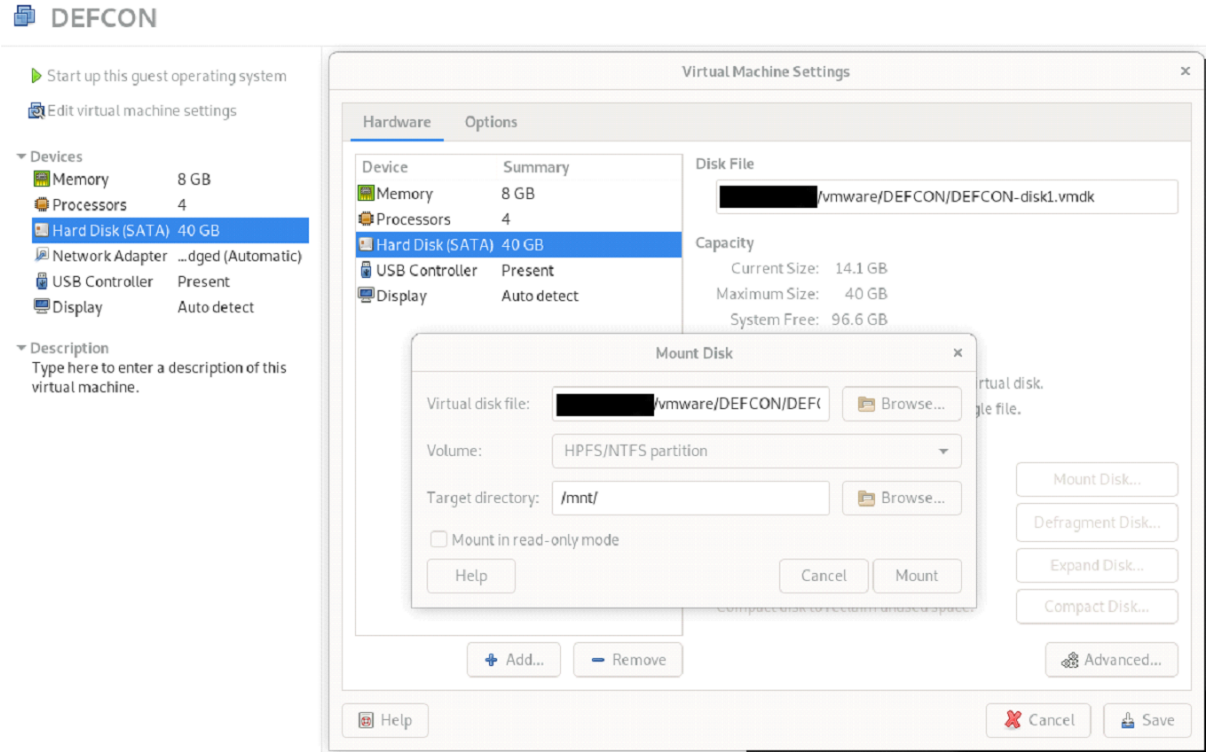


Figure 1 - vmdk mount

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

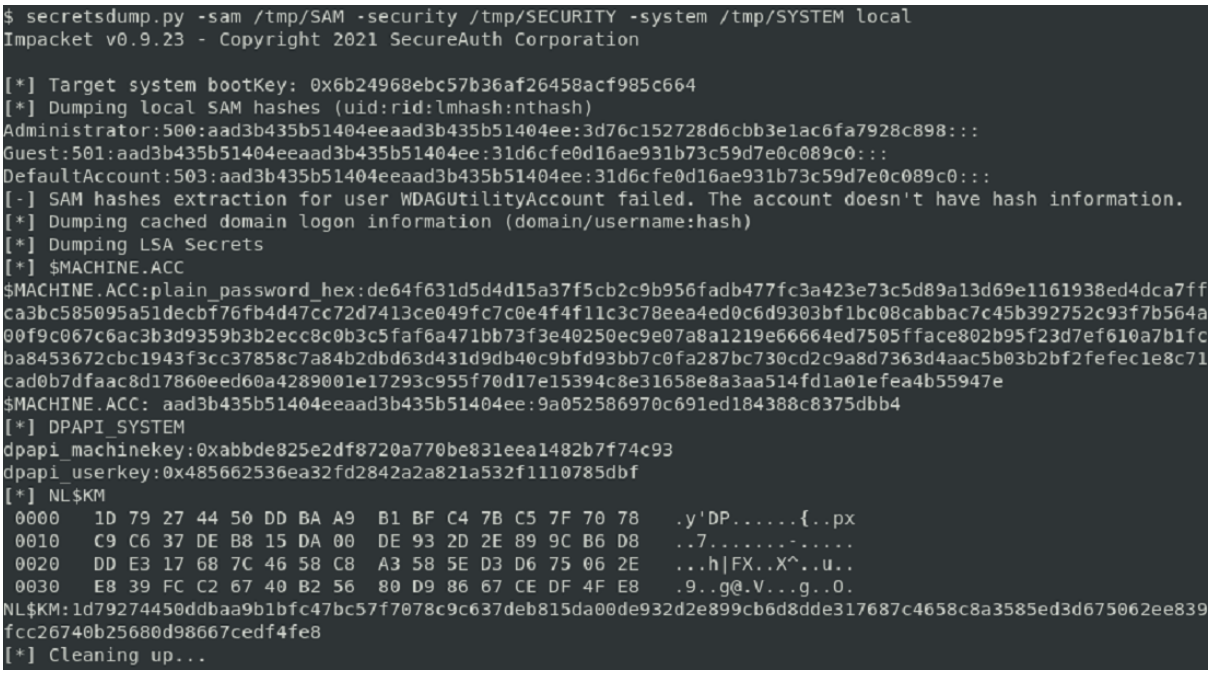


Figure 3 - retrieve local hashes with impacket/secretsdump

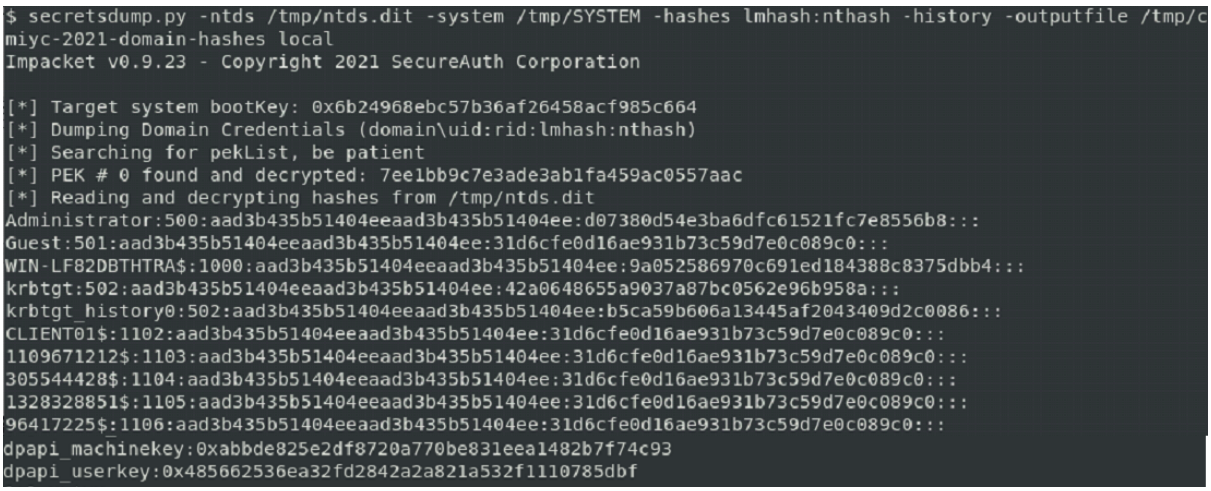


Figure 4 - retrieve domain hashes with impacket/secretsdump

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” :)

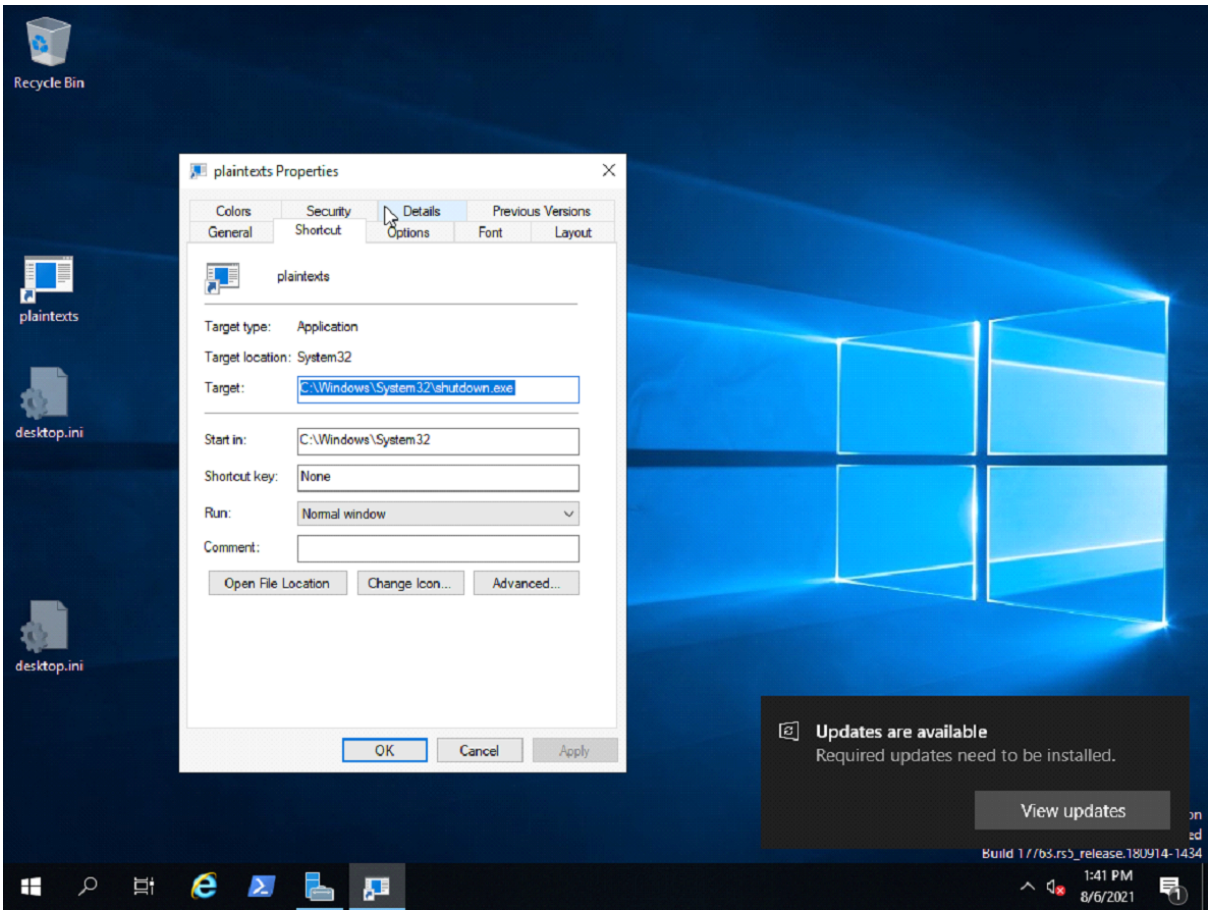


Figure 5 - plaintexts Properties

We proceed first by remove Windows Defender using Server Manager (from Manage/Remove Roles and Features). (Optionally just disable the protection.)

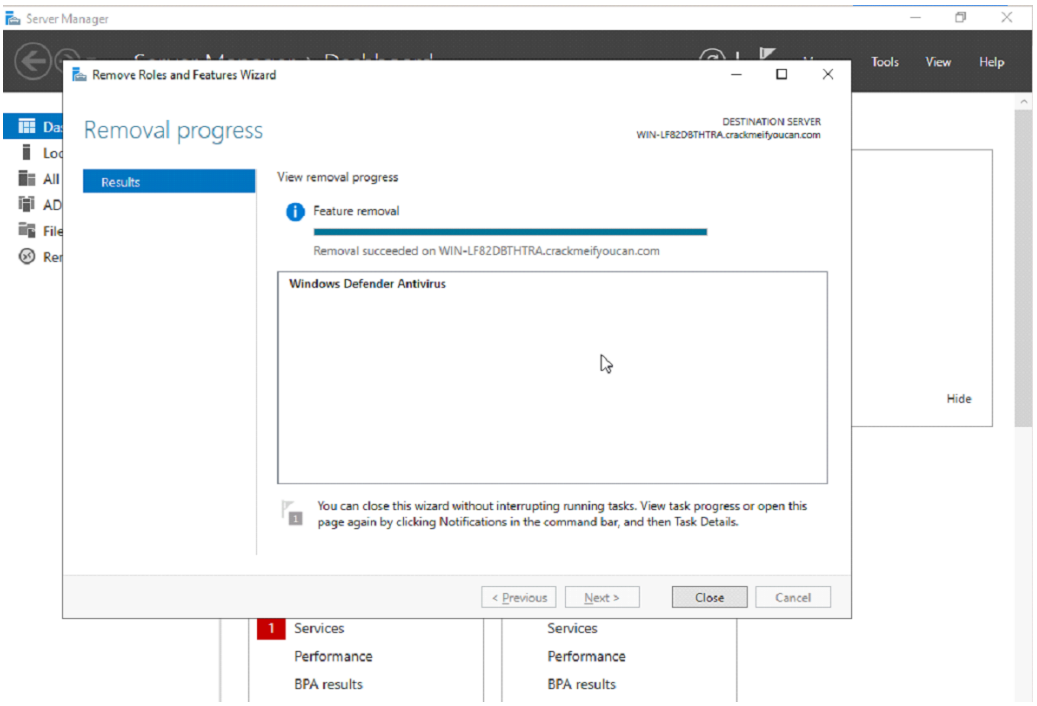


Figure 6 - remove Windows Defender Antivirus

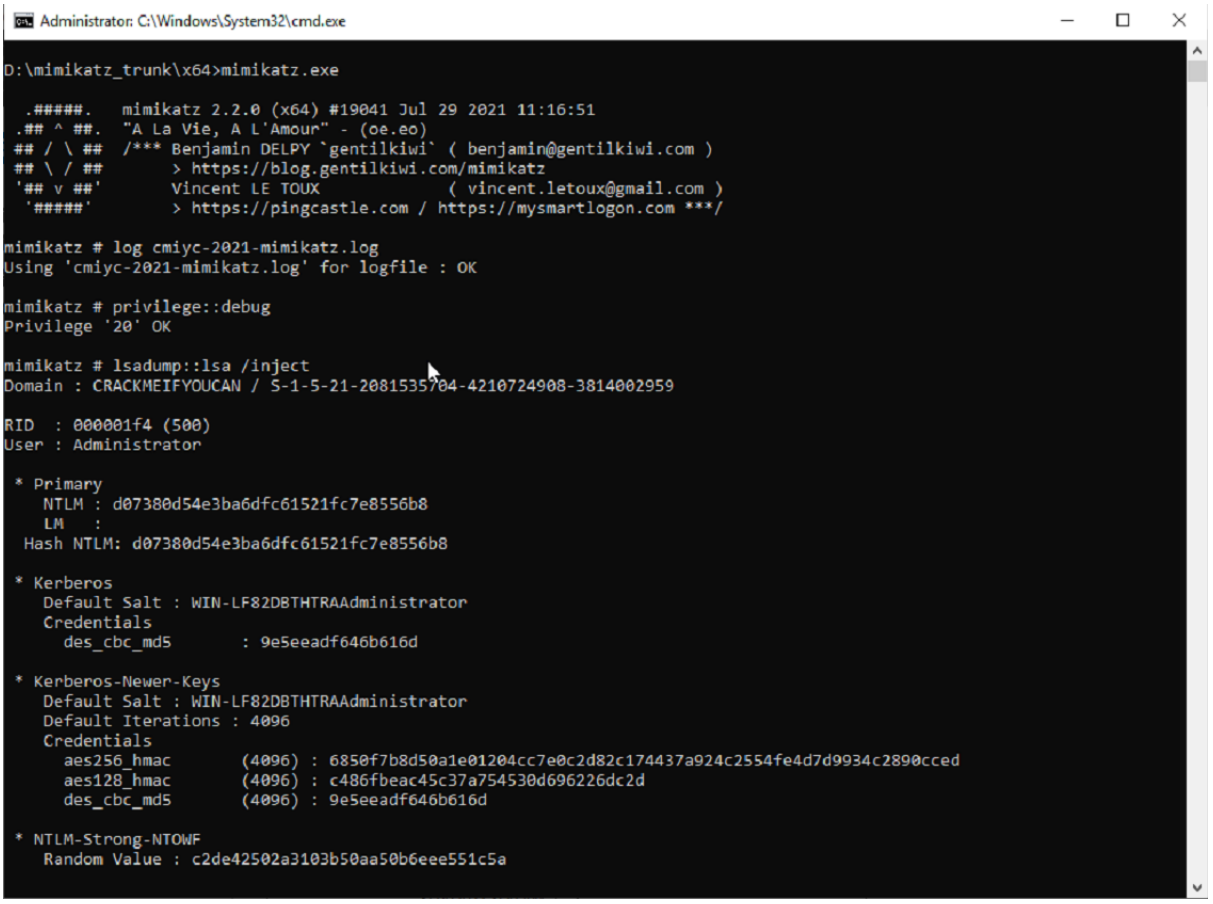


Figure 7 - retrieve credentials using lsadump with mimikatz

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”.

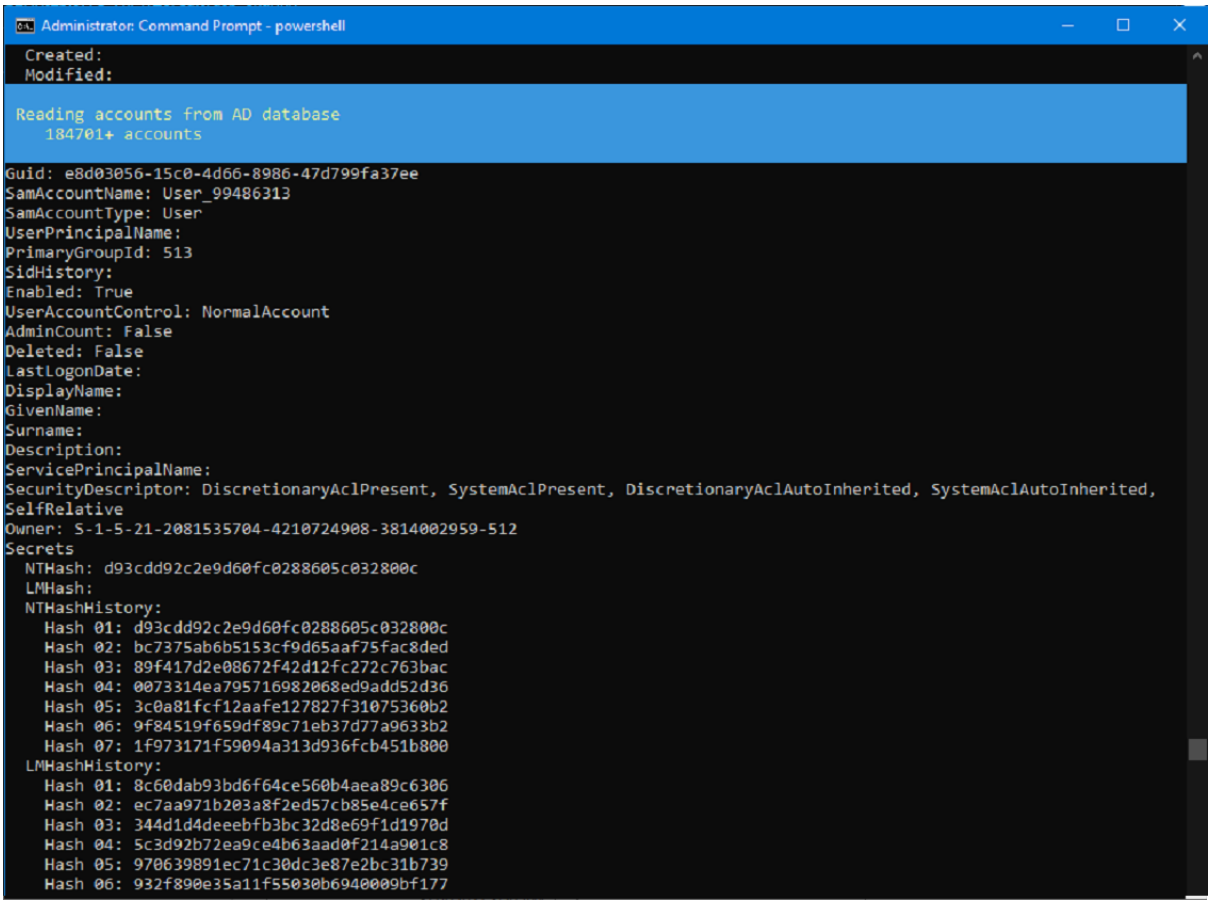


Figure 8 - dump accounts with secret attributes from AD using DSInternals Get-ADDBAccount

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.

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:.

history0 - 64 points
history1 - 32 points
history2 - 16 points
history3 - 8 points
history4 - 4 points
history5 - 2 points
history6 - 1 point

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.

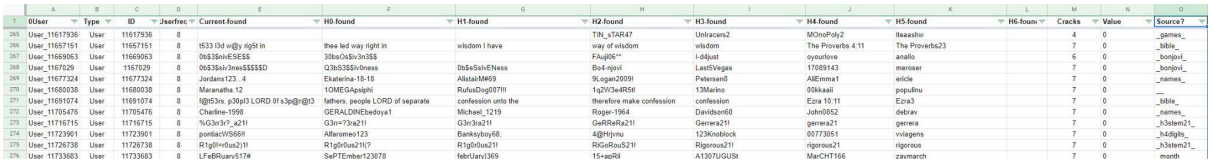
Unsurprisingly, History6 list did not give us any trouble. We basically managed to crack almost all of them instantly with standard attacks like rockyou+rules. The same goes for History5.

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.

History3 was the first layer that began to give us more of a challenge. 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) were 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	D	E	F	G	H	I	J	K	L	M	N	O
1	0User	Type	ID	Jserfrec	Current-found	H0-found	H1-found	H2-found	H3-found	H4-found	H5-found	H6-found	Cracks	Value	Source?
265	User_11617936	User	11617936	8				TIN_sTAR47	Unlracers2	MOnoPoly2	iteaashw		4	0	_games_
266	User_11657151	User	11657151	8	t533 !3d w@y rig5t in	thee led way right in	wisdom I have	way of wisdom	wisdom	The Proverbs 4:11	The Proverbs23		7	0	_bible_
267	User_11669063	User	11669063	8	0b3nlvESE$$	30bsOs$iv3n3$$		FAuj06**	I-d4just	oyourlove	anallo		6	0	_bonjovi_
268	User_1167029	User	1167029	8	0b3slv3ness$$$$$D	Q3b$3$$lv0ness	0b$eSslvENess	Bo4-njovi	Last5Vegas	17089143	meroser		7	0	_bonjovi_
269	User_11677324	User	11677324	8	Jordams123...4	Ekaterina-18-18	AlbstairM#69	9Logan2009!	Petersen8	AllEmma1	ericle		7	0	_names_
270	User_11680038	User	11680038	8	Maranatha.12	1OMEGApsiphi	RufusDog007!!	1q2W3e4R5!!	13Marino	00kkaaii	populinu		7	0	___
271	User_11691074	User	11691074	8	!@!53rs. p39pl3 LORD !9! s3p@r@t3	fathers. people LORD of separate	confession unto the	therefore make confession	confession	Ezra 10:11	Ezra3		7	0	_bible_
272	User_11705476	User	11705476	8	Charline-1998	GERALDINEbedoya1	Michael_1219	Roger-1964	Davidson60	John0852	debrav		7	0	_names_
273	User_11716715	User	11716715	8	%G3n3r?_a2!!	G3nr73va2!!	G3n3va2!!	Ga6RaRa21!	Gerrera21!	gerrera21	gerrera		7	0	_h3stem21_
274	User_11723901	User	11723901	8	pontiacVS66!!	Alfaromeo123	Banksyboy68.	4@Hjivnu	123Knoblock	00773051	vviagens		7	0	_h4digits_
275	User_11726738	User	11726738	8	R1g0l=r0us2)1!	R1g0r0us21!?	R1g0r0us21!	RiGeRouS21!	Rigorous21!	rigorous21	rigorous		7	0	_h3stem21_
276	User_11733683	User	11733683	8	LFeBRuarv517#	SePTEmber123078	febrUarv!369	15+apRil	A1307UGUSt	MarCHT166	zavmarch		7	0	month

(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.

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 a 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@sc<se3si1so0ss$sh5sv^sl!st+
sa4sc<se3si1so0ss$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 need 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 also does a unique sorting of the words per psalm to further reduce the number of combinations. It now contains essentially all of 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 can not 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 some regex-magic to untangle it.

Bonjovi

BON JOVI :D and variants on the word 'Obsessiveness'

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 random order	mislabel a product.1
History0	we didn't crack anything here	afraid be a mislabel to

(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&)

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 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

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	words that relate to the name + $d	what love would1
History2	2-3 words that relate 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	!0v3 w4n+ w0u!d wh4+ A5k

Due to the fact that 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.

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.

contest
DEF CON, Vegas, Ballys, etc etc etc

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

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 outbriefing referred to as the “intern-generated” hashes.

h2stem
h2 column has the stem word - always [word]2020 or 2021, with optional single special

Training
The plains found within the VM.
powershell: repair with “esenutil” dump with dsinternals (directory services internals) + filter to only have the user objects (remove the computer objets) + convert from UTF-16 to UTF-8.
The easiest way to extract the passwords was using “strings -e l” and then grep “PASSWORD_IS” on the output.

Website
Did not crack any - this puzzle totally escaped us. The information required had to be pulled from other metadata associated with specific users.

Other stuff we did

Autocrack for sleep time:

autocrack.sh

```
#!/bin/sh
while true; do
shuf -n 10 /root/dd/dict_all.dict.txt | /root/dd/smartquote.pl| sort -u |sed -e 's/^/1\t/'
> /root/dd/table
shuf -n 2 /root/dd/template.seed > /root/dd/template
```

```
perl /root/dd/simple_table.pl /root/dd/template /root/dd/table |
/root/git/hashcat/hashcat -m 1000 /root/dd/all.left -g 100000 -o /root/dd/current.out
--remove --generate-rules-func-max=30
/root/git/hashcat/hashcat -m 1000 /root/dd/all.left /root/dd/dict_all.dict.txt -g
10000000 -O -w3 -o /root/dd/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
```

“Blind” non pattern attacks based on all cracks:

```
-----START-----------------------quick and stupid 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
---------END----------quick and stupid sh--------------------
```

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]

Conclusion

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 9 – Final Scores

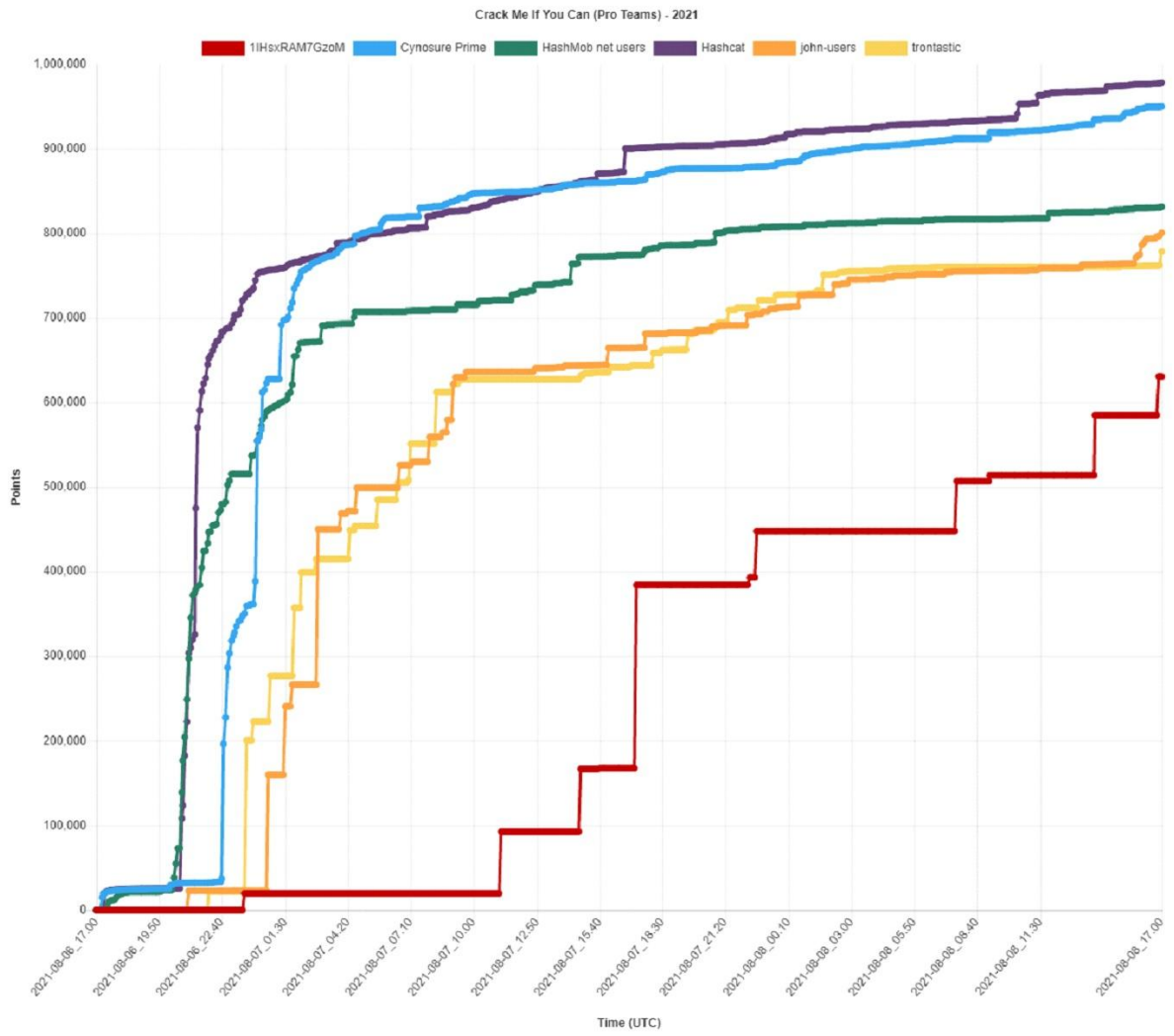


Figure 10 – Final Stats

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 most dull 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.