Team Hashcat write-up for Crack Me If You Can 2014

[Active Members]

alotdv	EvilMog	NullMode	ToXiC
atom	hashtka	philsmd	undeath
blandyuk	Hydraze	purehate	unix-ninja
BlowCane	K9	rurapenthe	Xanadrel
chancas	kontrast23	s3inlc	xmisery
dakykilla	legion	Szul	
dropdead	m3g9tr0n	The_Mechanic	

[Software used]

- List Condense (Hash-Management team software)
- hashcat
- oclHashcat
- Passcovery Suite (for challenge 3)
- JtR (for challenge 2 and challenge 4)
- Tons of small scripts to generate plains matching patterns

[Hardware used]

Most of us were available all the time. We can say we had around 15-20 members 16H/day working on it. We had tons of GPU, both AMD and some NVidias, but this time with much less problems regarding to driver and overheating since we were prepared carefully. However, in this contest, it was not the hardware power that matters. If you have such slow algorithms then the only way to crack multiples of them is with a 100% straight attack and with a set of password candidates that you really expect to hit.

[The first action to prepare the contest]

was to select a few members that would come up with ways to generate the plains Korelogic wanted us to submit. The main challenge was to not over complicate the resulting plains. At one point, we actually thought about dropping out of the contest because we thought it was impossible to generate plains that were difficult enough for the other teams to crack, yet still within the rules of the contest.

However, we persevered and came up with 31 different ways to generate plains. Only a third of which we considered “hard”, a third was “medium” and the last third was “easy”.

The base of almost all of our plains was a selection of the rockyou dictionary, which atom came up with.

Wordlist selection process:

1. We'll use rockyou, here's why:

- Makes minga happy, using a full known wordlist
- We will select only "rare" words from the wordlist, so that other teams will not quickly realize it's from rockyou
- It’s big (14 million entries, makes it hard on slow hashes even if other teams realize it was rockyou)
- Assures correct charset from Korelogic
- There is a count-version (see link below), so we can pick the lowest used words
- Picking lowest used words will steer other teams onto a false track, which our intentional goal is to poison their google searches, as this will lead them to the wrong dictionaries
- Results will have typos in it, so by mangling typo'd plains the chances are very high other teams "think" correctly typed word was used so they can't find the mangling algorithm

```
$ wget http://downloads.skullsecurity.org/passwords/rockyou-withcount.txt.bz2
$ bunzip2 rockyou-withcount.txt.bz2
$ dos2unix -f rockyou-withcount.txt
```

2. Sort out 8 bit chars

```
$ /root/hashcat-utils-1.1/req-exclude.bin 16 < rockyou-withcount.txt | sponge rockyou-withcount.txt
```

3. Randomly sort it, because original list is sorted alphabetically in 2nd instance

```
$ sort -R rockyou-withcount.txt | sponge rockyou-withcount.txt
```

4. Remove words that are "too long". The max allowed length is 12, so we use only words of max length 10

```
$ /root/hashcat-utils-1.1/len.bin 1 19 < rockyou-withcount.txt | sponge rockyou-withcount.txt
```

5. Select only words from low "occurrence" types used for poisoning

```
$ grep "      2 " rockyou-withcount.txt >  final
$ grep "      3 " rockyou-withcount.txt >> final
$ grep "      4 " rockyou-withcount.txt >> final
$ grep "      5 " rockyou-withcount.txt >> final
$ grep "      6 " rockyou-withcount.txt >> final
$ grep "      7 " rockyou-withcount.txt >> final
```

$ mv final rockyou-withcount.txt

6. Now it's safe to remove the counter

$ cut -b9- < rockyou-withcount.txt | sponge rockyou-withcount.txt

7. Remove user "mutations". This step is optional, but using only clean words gives us easier readable mutated results. This increases the chances that Korelogic will accept them as they are human readable

$ /root/hashcat-utils-1.1/req-exclude.bin 8 < rockyou-withcount.txt | sponge rockyou-withcount.txt
$ /root/hashcat-utils-1.1/req-exclude.bin 4 < rockyou-withcount.txt | sponge rockyou-withcount.txt
$ /root/hashcat-utils-1.1/req-exclude.bin 2 < rockyou-withcount.txt | sponge rockyou-withcount.txt

8. Now we ensure the remaining words are not just found in rockyou.txt in case the other teams grep for the words instead of using google

- It's better for us if they have multiple matches.
- To make this possible, we need to create a small tool that checks if the remaining words are in a different common wordlist
- To bring any sense into the next action, it's important that the other common wordlists are bigger than rockyou in case other teams select the wrong one

$ twodict.pl

9. Using the "twodict.pl" script, we make sure remaining words are also in other common wordlists as well:

$ perl twodict.pl wikipedia-wordlist-sraveau-20090325.txt rockyou-withcount.txt > pre-final

10. At this point, there are ~195k words left, example:

puala	kalongan	palay
gregw	spongecakes	whippo
cankles	jolibois	guding
zailor	boskoop	

11. One more round, just for safety

$ perl twodict.pl wikipedia-wordlist-sraveau-20121203.txt pre-final | sponge pre-final

12. At this point, there are ~91k words left, example:

cankles	boskoop	paltin
benado	hubsch	qaadir
garmisch	adoptada	whate
chemney	arester	aryeh
neata		

- There are some "hard" ones and some that seem to be usernames, etc.
- If we put in usernames it's easier for them to recognize the original list is rockyou, but usernames are
- mostly in english.
- So we will remove all words that include english substrings, names and so on, etc..
- To get there, some more work is needed:

13. Little perl script to remove lines that match substrings from a different list:

```
$ http://rlimatches.pl
```

- Running this perl script is extremely inefficient, so it will take ages; have patience!
- The output will be written to "rockyou-withcount.txt.left"
- And that's what it is, the stuff that survived the top10k passwords
- This is used to strongify the selection of the remaining plains

14. First we remove top10k passwords

```
$ head -10000 /root/dict/untouched/rockyou.txt > top10k.txt
$ perl rli-matches.pl top10k.txt
$ mv pre-final.left final
```

15. At this point, there are ~82k words left, example:

cankles	boskoop	paltin
benado	hubsch	qaadir

16. One more round, just for safety

```
$ mv pre-final.left pre-final
$ head -1000 /root/dict/untouched/facebook-first.txt > fb1k.txt
$ perl rli-matches.pl fb1k.txt
$ mv pre-final.left final
```

17. At this point, there are ~78k words left, example:

cankles	boskoop	benado
hubsch	qaadir	garmisch
adoptada	whate	chemney

18. A quick test shows that the remaining words are in many big common dictionaries:

$ grep ^adoptada$ *

rockyou.txt:adoptada
wikipedia-wordlist-sraveau-20090325.txt:adoptada
wikipedia-wordlist-sraveau-20121203.txt:adoptada

$ grep ^chemney$ *

facebook-last.txt:chemney
rockyou.txt:chemney
wikipedia-wordlist-sraveau-20090325.txt:chemney
wikipedia-wordlist-sraveau-20121203.txt:chemney

19. Split the result into 60 x 1k wordlists for further processing

$ split -l 1000 final

xaa, xab, etc are the final outfiles.

- Theoretically any of the outfiles are equally good, pick whichever you want.
- Searching them with google should result in crap, not in rockyou.
- If that's the case, we successfully poisoned the list, since we didn't modify anything to the original
- list, so minga can't complain. :)

All password generating code can be found here: https://hashcat.net/events/CMIYC2014/passgen/

The next step was to provide an overview and a platform to compile everything. I set up the google spreadsheet below mainly to keep track of the length distribution.

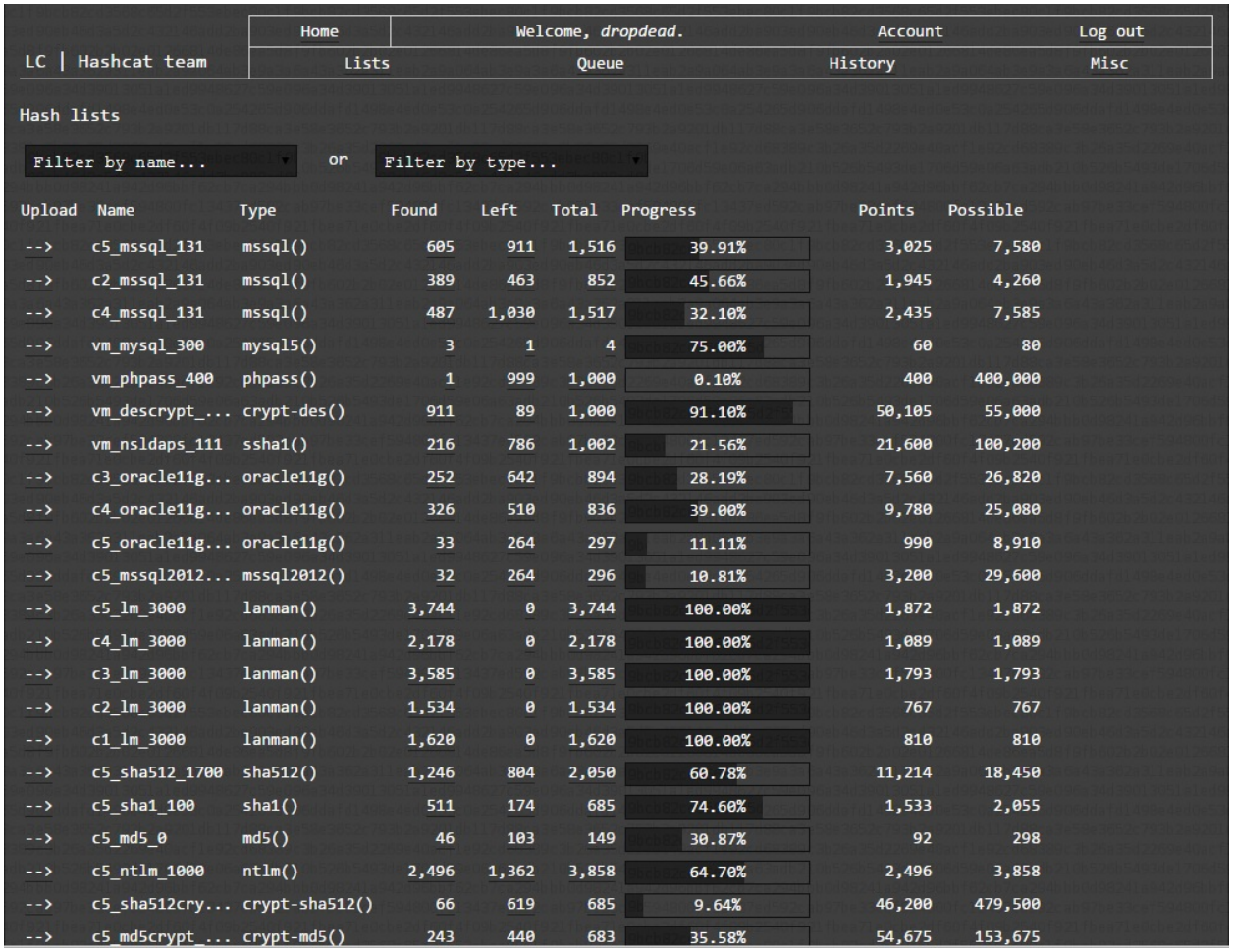
	Total Current Password Submissions	20000	20000	10000	3000	7000	60000	**Submited:**
	Total Required Password Submissions	20000	20000	10000	0	0	60000	31
	Missing	0	0	0	0	0	0	100.00%
name	**description**	**length 8**	**length 9**	**length 10**	**length 11**	**length 12**	**size**	**submitted**
atom-01	one input base words, appends first 4 byte of cryptographical weak digest encoded with KL charset	0	1000	1000	0	0	2000	TRUE
atom-03	one input base words, append sum of telephone countrycodes matching 2-char iso-3166 countrycode	0	0	2000	0	0	2000	TRUE
atom-04	one input base words, permutes the word x times. x = product of all ascii values of the chars in the word	0	0	0	0	1000	1000	TRUE
atom-05	one input base words, appends first 4 byte of md5 digest encoded with KL charset	0	1000	1000	0	0	2000	TRUE
atom-06	one input base words, mix-in first 4 byte of md5 digest encoded with KL charset	0	1000	1000	0	0	2000	TRUE
atom-07	same as atom-5 but encoded with charset limited to a-z only	0	0	0	0	1000	1000	TRUE
atom-08	same as atom-6 but encoded with charset limited to a-z only	0	0	0	0	1000	1000	TRUE
atom-10	one input base words, rotate consonants a->e, e->i, etc	0	0	0	0	1000	1000	TRUE
atom-11	one input base words, first char -1, second char +1, third char -1, ...	0	0	0	0	1000	1000	TRUE
atom-12	one input base words, simply mirror in the middle: password -> wordpass, 12345 -> 45312	0	0	0	0	1000	1000	TRUE
atom-13	one input base words, us keyboard vertical replace, 1 becomes q, q becomes a, a becomes z, etc	0	0	1000	0	0	1000	TRUE
atom-15	two input base words, use first 3 char from each and join, then apply rule "TX $Y $Z" where X=[012345], Y=?d, Z=?s	2000	0	0	0	0	2000	TRUE
atom-16	two input base words, filter in both char by char, then apply rule "TX ^Y $Y" where X=[1234567], Y=?d	0	2000	0	0	0	2000	TRUE
blandyuk-4	]] oXY iXY iXY TN r	600	600	400	400	0	2000	TRUE
blandyuk-7	[{{ iXY iXY TN TN TN	600	600	400	400	0	2000	TRUE
blandyuk-8	u iXY iXY iXY TN	600	600	400	200	200	2000	TRUE
blandyuk-9	{{{ so! se$ si3 TN	600	600	400	400	0	2000	TRUE
blandyuk-A	f]] sa^ su+ TN TN	600	600	400	200	200	2000	TRUE
dropdead-1	prepend, append random chars	1000	1000	0	0	0	2000	TRUE
dropdead-2	insert random chars after pos 2 based on 4 char words (xmisery-7)	1000	1000	0	0	0	2000	TRUE
dropdead-3	insert random chars after pos 3 based on 5 char words	1000	1000	0	0	0	2000	TRUE
dropdead-5	Forex Symbols with m1 and d1 timeframe	2000	0	0	0	0	2000	TRUE
k9-1	ceasar with substr4	967	1033	0	0	0	2000	TRUE
xmisery-1	ROT47	600	600	400	400	0	2000	TRUE

xmisery-2	ROT47 Reversed	600	600	400	400	0	2000	TRUE
xmisery-3	ROT13 Reversed + Pattern (digit,digit,special,special)	600	600	400	200	200	2000	TRUE
xmisery-5	Pattern Prepend (special,special,special,special,digit,digit) + Word	0	200	400	200	200	1000	TRUE
xmisery-7	Pattern Insertion Wo + (special,special,special,special) + rd	600	600	400	200	200	2000	TRUE
undeath-1	scraped from anisearch.com	417	583	0	0	0	1000	TRUE
dropdead-6	?a?aatom?a?a (variable before and after) with toggle cased "atom" baseword	3216	4784	0	0	0	8000	TRUE
unixninja-1	Baseword "star" tmesis against webster's dictionary + leetspeak and toggle rule	3000	0	0	0	0	3000	TRUE

[The second main part of our preparations]

was to bring new members to the team. Our headhunter was blandyuk, who did a great job bringing amazing new talents to our team.

For our collaboration during the contest we used the Hash-Exchange-System LC again, which we had been using at the previous contests and has been greatly improved and maintained by Xanadrel.



LC | Hashcat team

Home | Welcome, *dropdead*. | Account | Log out

Lists | Queue | History | Misc

Hash lists

Filter by name... or Filter by type...

Upload	Name	Type	Found	Left	Total	Progress	Points	Possible
-->	c5_mssql_131	mssql()	605	911	1,516	39.91%	3,025	7,580
-->	c2_mssql_131	mssql()	389	463	852	45.66%	1,945	4,260
-->	c4_mssql_131	mssql()	487	1,030	1,517	32.10%	2,435	7,585
-->	vm_mysql_300	mysql5()	3	1	4	75.00%	60	80
-->	vm_phpass_400	phpass()	1	999	1,000	0.10%	400	400,000
-->	vm_descrypt_...	crypt-des()	911	89	1,000	91.10%	50,105	55,000
-->	vm_nsldaps_111	ssha1()	216	786	1,002	21.56%	21,600	100,200
-->	c3_oracle11g...	oracle11g()	252	642	894	28.19%	7,560	26,820
-->	c4_oracle11g...	oracle11g()	326	510	836	39.00%	9,780	25,080
-->	c5_oracle11g...	oracle11g()	33	264	297	11.11%	990	8,910
-->	c5_mssql2012...	mssql2012()	32	264	296	10.81%	3,200	29,600
-->	c5_lm_3000	lanman()	3,744	0	3,744	100.00%	1,872	1,872
-->	c4_lm_3000	lanman()	2,178	0	2,178	100.00%	1,089	1,089
-->	c3_lm_3000	lanman()	3,585	0	3,585	100.00%	1,793	1,793
-->	c2_lm_3000	lanman()	1,534	0	1,534	100.00%	767	767
-->	c1_lm_3000	lanman()	1,620	0	1,620	100.00%	810	810
-->	c5_sha512_1700	sha512()	1,246	804	2,050	60.78%	11,214	18,450
-->	c5_sha1_100	sha1()	511	174	685	74.60%	1,533	2,055
-->	c5_md5_0	md5()	46	103	149	30.87%	92	298
-->	c5_ntlm_1000	ntlm()	2,496	1,362	3,858	64.70%	2,496	3,858
-->	c5_sha512cry...	crypt-sha512()	66	619	685	9.64%	46,200	479,500
-->	c5_md5crypt_...	crypt-md5()	243	440	683	35.58%	54,675	153,675

Our two main ways of communication this time were an IRC-Channel and another Google spreadsheet.

[In advance of the contest]

we decided to divide our members into 4 groups and assign them algorithms to work on. This soon turned out to be unproductive since Korelogic decided to group their plains up into 5 “companies” and divide the pool of plains into 13-19 algorithms. A few hours into the contest, we therefore decided to change the assignments of our groups to align more accordingly to Korelogic’s companies and challenges. The assignments changed again during the second half of the contest several times. The main purpose of this was to mix up the analysis of patterns that everyone did.

Group	**1**	**2**	**3**	**4**
Count	8	7	7	7
	#teamchannel1	**#teamchannel2**	**#teamchannel3**	**#teamchannel4**

Members	BlowCane	atom	alotdv	chancas
	coolbry95	blaz	blandyuk	dakykilla
	epixoip	legion	dropdead	Hydraze
	evilmog	NullMode	hashtka	K9
	kontrast23	s3in!c	philsmd	purehate
	rurapenthe	xan	Szul	Rolf
	ToXiC	xmisery	The_Mechanic	undeath
	unix-ninja			
Assigned Algos				
(will	Company1	Challenge3	Company3	Company4
change	Company5	Company2	Challenge8	vm_mysql
during	vm_ssha	vm_phpass_400	Challenge5	vm_des
contest)	vm_shadow_1800	Challenge4	Challenge6	

[Contest]

Whenever a contest starts, it takes a few hours to find out what hash-types are used, in which format they are, etc., etc. So that was our first action before we started cracking.

With these screenshots you can see the company specific hash-types identified. Note that not all hash-types can be listed as not all hash-types are supported by our system.

https://hashcat.net/events/CMIYC2014/pics/result_c1c2.png
https://hashcat.net/events/CMIYC2014/pics/result_c3c4.png
https://hashcat.net/events/CMIYC2014/pics/result_c5vm.png

Also note, these are screenshots from after the contest, that is why there are results listed.

Once we finished with Free-For-All (FFA) cracking, we began with pattern analysis. This time we spent nearly all of our time on this part as from experience with previous contests it turned out this is the key to win a contest.

Here's the pattern we noticed during the entire contest

Global pattern: https://hashcat.net/events/CMIYC2014/pics/pattern_global.png
Company pattern: https://hashcat.net/events/CMIYC2014/pics/pattern_companies.png

[WPA]

The WPA's found in the companies 2, 3, 4 and 5 at position 14 were in JtR format. That's a bit of a problem as there exists no converter from JtR format to any other format, we would have been forced to use JtR for cracking those WPA.

I think that was a bit a bad decision from KoreLogic. It would have been OK if they simply generated network .cap files. In that case all WPA crackers would have been able to load it.

What I had to do is to patch JtR a bit. Good thing is that JtR internally uses hashcat c-types for WPA cracking. All I had to do was to store the hccap structure once it was loaded by JtR and write it to a file so that hashcat was able to load it:

This patch to wpapsk.h from JtR did it:

[code]
FILE *fd = fopen ("lala.hccap", "ab");
// Debug_hccap();
fwrite (&hccap, sizeof (hccap), 1, fd);
fclose (fd);
[/code]

After that, I simply ran JtR with the KoreLogic format and all handshakes were stored to lala.hccap, per company. The result are 4 big .hccap files, but it's no problem as hashcat can load multiple handshakes at once. Results can be found here:
https://hashcat.net/events/CMIYC2014/hccap/

[Challenge2]

As no hashcat software supports cracking of .docx files, we had to use JtR to crack them. The JtR version used is the one maintained at magnums github tree.

For the pattern, it was obvious they were all mangled version of the basewords:

- 123456 (?)
- password
- microsoft

There must be more basewords, but we didn't manage to identify them in time. This algo is so damn slow!

Here's how we organized the mask based cracking:
https://hashcat.net/events/CMIYC2014/pics/challenge2_masks.png

[Challenge3]

Not much to mention here as all passwords were simple digits. @KoreLogic: was this a result of a banned plain from another pro-team?

Again, as hashcat does not support cracking of .doc files, we had to use JtR to crack them. The only problem here was that the files were unable to be cracked with JtR for some unknown reason. They simply did not crack even when giving the correct password. Due to the time pressure I didn't try to fix the problem.

I simply switched to Passcovery to crack them. That was a bit of a problem too, as Passcovery does not allow multi-hash cracking for .doc files. So I wrote a simple batch-script and called it 1000 times :)

[Challenge4]

Same as Challenge3, ran some dictionaries against it and found patterns based on financial words such as “billing”, “expense”, “mastercard”, “credit”, etc. and then produced wordlists based on those basewords discovered.

[Challenge5]

Some of us tried to load the rars into different rar-cracking tools but since it was rarv5 which we figured out afterwards no tool supported it. We managed to convert the hashes with john and load them into john but failed to crack any of them.

[Challenge6]

Same as Challenge5.

[Challenge7]

Challenge 7 appeared to some of the members of team hashcat as a very interesting Challenge. We quickly found out that this only could be Cisco's new scrypt hash type (the other new one that we did see in the wild is type 8 which according to some cisco documents seems to be using PBKDF2-SHA256).

We tried to crack the Challenge 7 hashes with the scrypt configuration N = 16384, r = 1, p = 1 . The base64 table for the encoding/decoding part that we tried to use is;
./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz .

Unfortunately, we did not crack any hash in a reasonable amount of time. But after we saw this tweet from Korelogic (https://twitter.com/CrackMeIfYouCan/status/497809778580533248) that Challenge 7 was now disabled, we completely moved on to crack other Challenges/hashes and did not try further to understand that hash format, nor did we try to crack the hashes anymore.

[VM exploration]

For the virtual machine given in a .ova file, we already knew the password for LUKS from KoreLogic (http://contest-2014.korelogic.com/VM_Challenge_notes.txt), but after that, we also needed the root and/or cmiyc user's password. Of course there could be many ways to get to the /etc/shadow file etc., but we decided to just start an Ubuntu live cd in .iso format that we had handy and mounted the encrypted drive from the live cd. After that, we were able to dump /etc/shadow, modify it to change the root user's sha512crypt hash and look around at other interesting files etc. Afterwards, we rebooted and logged in w/ the modified root password.

From then on we were able to find several hints about passwords e.g. within text files starting from the root folder / . Furthermore, with those hints and some configuration files, we quickly identified some patterns like "wppassword123!@#", "cmiycpassword123!@#" etc. We then went on and were able to decrypt the ecryptfs home folder of the cmiyc user. Unfortunately, we didn't find many hashes there, as we were initially expecting to find under that encrypted cmiyc folder. In the end, we can't complain, as the virtual machine did provide us with several hash lists easily obtained, including mysql 4/5, phpass, descrypt, ldaps/ssha and sha512crypt hashes.

[Management during the contest]

was mainly to keep in contact with Korelogic, submit our finds, and fix our problems which we had a lot of. First off, there was a problem submitting the doc and docx plains. Although Korelogic initially explained on their FAQ a format that suggested to submit them as “filename plain”, this did not work. Throughout the first half of the contest, Korelogic’s DoS-filter kicked in and kept us from submitting and kept our users from viewing the overall contest statistics.

Another big problem for us was the submission of the LM hashes. Since we had to split them into two halves, we were uncertain why we only had 9000 registered cracks, but almost 12000 founds, while other teams had 10000 to 11000 registered cracked. One friendly guy at Korelogic tried to debug this with us, but the contest ended without a solution.

Another problem was the submission of DES, SHA512, mssql and vbulletin hashes, which Korelogic eventually managed to fix.

Additionally, there were hash types/lists which Korelogic’s system did not care about at all, like the vm_mysql, but were later happy to include.

The “scrypt” subject is a little controversial, since we spent many hours trying to crack in addition to all of the hard work that atom spent to figure out the correct settings for how the hashes had been produced. Korelogic later just removed these hashes. The scrypt value of 9999 was too high to allow for dupes, which some street teams were receiving credits for after cracking other hash types. On the other hand, with Korelogic just removing these hashes altogether from the contest was also disappointing, considering all of the energy that our team put into this which was now lost time and energy. We would have rather seen just a reduction of the value.

In contrast to our feeling on scrypt, we were happy to see the removal of the grub2 hashes after our inquiry into Korelogic that its’ salt were bogus.

[Thoughts about the contest]

are pretty good despite the initial doubts about the new precontest methods. The process of generating plains for the other teams to crack, while staying within the rules of Korelogic to keep things simple, was painful. It was very unsettling to not know which of our plains would be rejected by penalty and replaced by all digits, and which ones would make it. Despite all of our initial doubts about the teams creating the plains, it was a very pleasurable experience since the overall plains were structured in a way that we managed to keep finding them, which was not the case in other contests. Also, we were very happy communicating with Korelogic's support throughout the duration of the contest. They took great effort into solving every problem that we presented to them.

[Thanks]

Ty Fox :)