Team Hashcat Contest-Writeup

Congratulation to every team that participated especially to john-users for giving us the thrills all the way to the end and we welcome CynoSure_Prime for building a new competitive team!

Preparation

When the contest description was available and it said that there will be GOST (2012) and PHC finalists, amongst others, we decided to write special dedicated (non hashcat) crackers for them.

The problem was that we didn't know which PHC finalist algorithms would be selected to be used in Hashrunner. Because of this, we wrote a generic PHC finalist cracker. It reads password candidates from stdin and passes them to specific PHC finalist functions. We created a multi-threaded application, with the ability to crack multiple hash types and run on multiple operating systems. The final application was basically a mini-hashcat just for the PHC finalists.

We also developed a cracker for the GOST 34-11-2012. We did this, however, for the real GOST reference implementation which differs from the stricat version, which uses a GOST-2012 version called Streebog. Due to the time constraints in the competition, we did not look into the specific details of the differences between them.

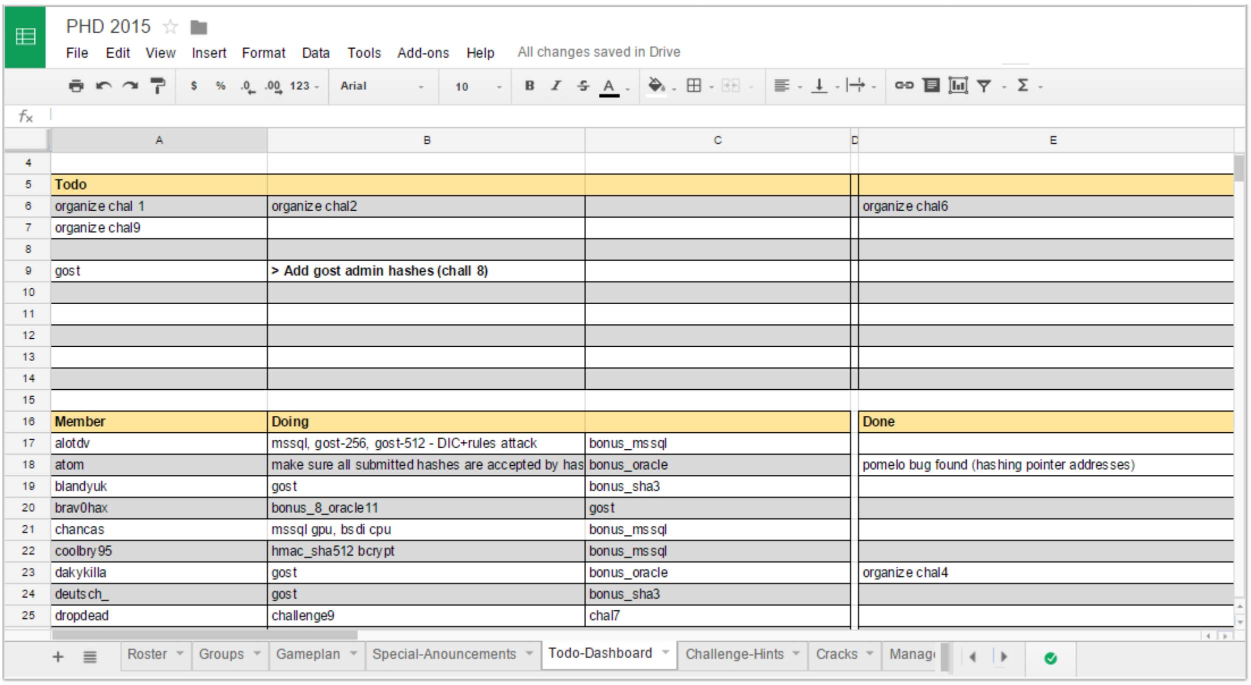
Due to the increasing requirement for brain power in these competitions, the team has grown to 31 members. But from those 31 members, only 20 were active. For example, the total number of cracked hashes sent in by the 11 least active members were less than the total number of cracked hashes from every other member alone. During the contest, the members were split up into 4 groups, and were assigned specific tasks/algorithms. In addition to this we introduced a SCRUM-like task assignment methodology.

To mitigate the challenging task of information sharing and collaboration we used our known tool list-exchange (which we mentioned in previous contests), multiple IRC channels, a Teamspeak Server and Google Drive.

Team members

abaco	dakykilla	legion	Rolf	unix-ninja
alotdv	deutsch_	m3g9tr0n	rurapenthe	Xanadrel
atom	dropdead	minga	Szul	xmisery
blandyuk	epixoip	NullMode	The_Mechanic	
J0hnnyBrav0	evilmog	philsmd	tehnlulz	
chancas	Hydraze	purehate	ToXiC	
coolbry95	kontrast23	radix	undeath	

Spreadsheet for Information Sharing and coordination



	A	B	C	D	E
4					
5	**Todo**				
6	organize chal 1	organize chal2			organize chal6
7	organize chal9				
8					
9	gost	**> Add gost admin hashes (chall 8)**			
10					
11					
12					
13					
14					
15					
16	**Member**	**Doing**			**Done**
17	alotdv	mssql, gost-256, gost-512 - DIC+rules attack	bonus_mssql		
18	atom	make sure all submitted hashes are accepted by has	bonus_oracle		pomelo bug found (hashing pointer addresses)
19	blandyuk	gost	bonus_sha3		
20	brav0hax	bonus_8_oracle11	gost		
21	chancas	mssql gpu, bsdi cpu	bonus_mssql		
22	coolbry95	hmac_sha512 bcrypt	bonus_mssql		
23	dakykilla	gost	bonus_oracle		organize chal4
24	deutsch_	gost	bonus_sha3		
25	dropdead	challenge9	chal7		

Contest antics

We realized that the main focus of the contest was on the **GOST-2012** algorithm. While it was worth the most points, it was also unsalted, meaning potentially easy points. Therefore, we decided after the contest had already started, to develop a pure and native GPU implementation of GOST-2012 (Streebog). The speed for both variants (256 and 512 bits) on oclHashcat v1.37 is around 20MH/s on a single hd7970.

For the **Invuln** algorithm, without looking too deep into the salt-generation function (which we should have done in more detail), the algorithm was designed so that the salt generator (which depends on the password) itself returned something like a hash. In other words, the password creates a unique salt which we can then use to exploit it as an early-skip-out key of the loop by checking if it exists in the hash database. That can be done before running the real sha512crypt calculation which is the much more costly calculation. Also, by querying the database for the salt-key, this also enables us to select only those sha512crypt hashes that match this salt. The hashlist thus effectively becomes saltless as we do not have to check each candidate against each salt.
In the night between sunday and monday one of us decided to write a cracker attacking the weak salts. The cracker worked great except that due to its use as the salt in sha512crypt it was a bit truncated and gave "only" the first 12 bytes of the plaintext, but that was good enough and yielded another 250 real cracks in the end.

We decided not to put much effort into the **hashcoin** challenge because of the low starting value and the unknown coefficient which, after some testing, showed not to be very high. We wrote a multi-threaded cracker around the provided python script which was fairly slow and only yielded 250 cracks in the last 24 hours of the contest.

SCRYPT was indeed quite easy. We over thought the hint and tried to apply “real world” methodology. Random specials were stripped from the dictionary created, and run against the hashes. Due to a bug in CPU hashcat, we were unable to crack any of these for the first two days. After that point, we used JTR to run these in hopes the same bug did not exist. Unfortunately, it did not occur to us at that point to re-run the initially created dictionary with specials stripped. It would have been obvious we were on the right track having used:

```
sed -e ‘s/ /\n/g’ dolphin_hint > dolphin.dict
```

We opted to run simple 5 letter words and got two hits, “about” and “share”. We then switched back to the proper dictionary and cracked 100%. However, not all hashes were able to be loaded. Only 191 of 203 were accepted, leaving us to fill in the blanks. The organizers were clever in how they sorted the hash list (each hash was properly positioned to match the words in the song). This allowed us to fill in the blanks for the 12 hashes that were not able to be loaded.

Bugs; caused by organizers

GOST-2012

When the contest started it took us around half a day to realize that the Streebog version of GOST-2012 is very different to the reference GOST-2012 implementation. It was since this algorithm was very new to us, nobody also knew that there are two different versions of it. When we did not crack any of the GOST-2012 hashes we first expected them to be just very hard plains because of the high point value and not to be a different algorithm implementation. Then rewriting the dedicated GOST-2012 cracker took us another half a day. This could be avoided if PHD organizers added a note by indicating they were planning to use this special version of GOST-2012 and not the official reference version.

Pomelo

The major flaw (besides the same salt being used) was that the hash-generator developed by the PHD organizers hashed the pointer address to some word from a wordlist, but then it also used the correct password-length of that word. This made the input more or less unpredictable and therefore these hashes were impossible to crack. One idea could be to try to bruteforce the first 4 bytes only because chances are high that the original wordlist should also contain words of length 4. But we ignored that since such a bruteforce would probably include null-bytes and those would be hard to submit to the Hashrunner site.

Wonderful

We tried to run the provided PHP scripts with different versions of PHP but unfortunately all of them reported some code warnings (index out of bound bugs, implode() parameter type warning etc). When we tried to use the .php files AS-IS (i.e. provide some passwords for a registered user while ignoring the PHP warnings) several output errors were shown because of the aforementioned problems. It wasn't clear if we were supposed to go ahead and "fix" the scripts or if we should just ignore the false positives instead.

Something very helpful in that case, was that we could easily see which specific hash in the hash database required the use of one problematic functions (because the mode - first byte of "hash" - provided this information) and as a result we were able to separate the hashes into categories, i.e. md5($pass.$salt), md5($salt.$pass), sha512($salt.$pass), sha512($pass.$salt) hashes etc…

Once we had these new lists ready, we were able to use oclHashcat to crack the hashes for some of the lists (without the need of PHP etc). This strategy was successful and gave as a few cracks, but we still didn't completely understand (if we need and) how to solve the problem with, especially the do_xor () and permute () functions.

What is also interesting is that we discovered the existence of 974 weak hashes (i.e. hashes generated from the empty string - zero length - password), this would sum up to 974 * 15 = 14610 more points. For example, md5 ("") = d41d8cd98f00b204e9800998ecf8427e and one sample db.inc hash was md5:40481:52d41d8cd98f00b204e9800998ecf8427e (note that the digest, last 32 hex characters are identical, also note there were several of those raw md5 weak hashes).

Unfortunately, the uploading of these "cracks" (974 weak hashes) didn't seem to work since uploading with an empty password was not correctly verified by the Hashrunner web site.
This meant because there were so many weak hashes, the PHP script should be used AS-IS (no code corrections to remove the warnings/notices should be done by the contestants). But the main question which is still unanswered for us is: how could we get these points if those cracks were not accepted? We still do not know if the Hashrunner team required us to upload the "correct" user password (i.e. the character sequence used by the user before it was "converted" to the empty string). However, this would have been literally impossible, because in this case, there is no way to verify what the original password was (before the faulty php code lines corrupted it such that the empty string was used).

Bugs; caused by tools/software we used

- Pufferfish: The reference code for Pufferfish is simply not thread-safe (at least the one version which was used during the contest). epixoip (the author of Pufferfish) fixed this issue by writing an optimized version of the algorithm.

- oclHashcat: In the early beginning of the contest, we exploited the hashing mode -m 7200 (grub2) to crack the PBKDF2-HMAC-SHA512 but we ran into a bug with salt sorting. The problem was that we did not expect to have two random salts being the same but having a different iteration count. In reality, this wouldn't happen, but unfortunately it appeared in this artificial scenario. While fixing this issue we added native support for PBKDF2-HMAC-SHA512.

- hashcat: There was a bug in the SCRYPT implementation of hashcat which only occurs when the P value is >= 4. All hashes used in this contest had a P value of 16 which is totally unrealistic for a real-life server as you would effectively DoS the server by letting users login to it :) However, this bug's fixed in the latest version of hashcat.

Algorithms added to oclHashcat during contest

- Desc: Added new hash mode -m 11700 = GOST R 34.11-2012 (Streebog) 256-bit
- Desc: Added new hash mode -m 11800 = GOST R 34.11-2012 (Streebog) 512-bit
- Desc: Added new hash mode -m 11900 = PBKDF2-HMAC-MD5
- Desc: Added new hash mode -m 12000 = PBKDF2-HMAC-SHA1
- Desc: Added new hash mode -m 12100 = PBKDF2-HMAC-SHA512

Note, we already had PBKDF2-HMAC-SHA256 (-m 10900) which is why we did not need to add it.

The hash cracking itself

After the first 6 hours of extracting, sorting, and uploading every list to our list-exchange, the cracking process begun. The high level methodology was as follows:

- Run wordlist attacks, mask attacks, wordlists and rules
- Look at plains found
- Search for patterns and common sources
- Attack and repeat

The passwords appeared to be based primarily around Italian, Chinese and Norwegian dictionary/wordlists. Patterns identified were word duplication, Case Toggle, L33t, and special characters prepended, and appended (with some of them doubled afterwards).

Wishes for next contests

We kind of missed the hints (like tomato), and we would like to see a bonus system that rewards the most cracks in each type of hash, maybe up to top 3 teams in each type. The reason is that if there’s a situation where an algorithm’s point value is very high compared to the computational effort (GOST for example), a team can exploit this by using all their resources against that hash type and completely ignoring the others. Teams attempting to look at the majority of hash types would need to focus more time (and cracking power), meaning their efforts would not be rewarded as much as if they were to focus purely on the hash type giving the best points/time ratio. The organizers can not “downgrade” the points while the contest is running but rewarding teams which crack the most of each hash type (as seen in CMIYC) might be an effective mitigation against such a tactic.

For example, the team with the most cracks of a specific hash type gets 100 points, but teams with the second and third most cracks would get 50 and 20 points respectively. This encourages teams for taking the time to look at all hash types because, for each algorithm, bonus points will be rewarded.

Another suggestion for the upcoming contest in 2016 (and maybe other contests - yes you CMIYC? - could think about this idea too) that came to our mind is that there should be some kind of rewards for teams that keep staying at the first place for some time. For instance, the system should count the minutes that a team holds the lead and show this real-time value (in minutes and points) clearly on the webpage of the contest. In this way, teams wouldn't try to hold back cracks and surprise the other teams with huge uploads within the last minutes of the contest (this may not have been the case this year, but clearly it was during the previous years). The extra points per minute a team leads should of course not be exaggerated too, but an incentive that teams upload as quickly as they can.

With such a bonus-points system in place (for instance one that rewards extra points for the two cases mentioned above) it is important that contestants understand where exactly the (extra) points come from. That means that the contest web page should list this information clearly and possibly in real-time.

One final suggestion would be to have at least one Top 10 password for each algorithm, or a sample ciphertext:plaintext pair for each new algorithm, so that we may verify that our implementation of an algorithm is correct. This is important for a contest which requires coding.

The good, the bad, and the ugly

This contest was pretty much a coding challenge. Some like this, some don't. For us, we can't exactly decide. The effort put into the challenges was quite high and noticeable. Every new pattern that emerged in the plains or in the coding was a "you bastards"-moment.
This time the ugly girl in the corner was the Hashrunner site and its backend, especially after finding out that some "serious security flaws" (or call it content caching issue?) were discovered.

Once again, a big thank you to the Hashrunner team for making another fun competition!

Official Soundtrack: https://www.youtube.com/watch?v=Ktbhw0v186Q