Team Hashcat
writeup for
Crack Me If You Can 2017

Active members (21)

alotdv	atom	BlandyUK	Chick3nman
chancas	coolbry95	dropdead	hydraze
kontrast23	m3g9tr0n	N|GHT5	philsmd
Rolf	rurapenthe	Szul	ToXiC
TychoTithonus	undeath	unix-ninja	Xanadrel
xmisery			

The following team members provided additional GPUs to the team's Hashtopussy instance:

- abaco
- EvilMog
- franky
- NSAKEY
- NsN

Contest organization

This year, the actual running of the contest was colocated with DerbyCon instead of Defcon. The contest officially started on September 22nd at 5:00 AM EDT / 9:00 AM UTC, running for 50 hours.

The organization was a lot easier this time for KoreLogic, because they did not have to maintain a scoreboard or a system for uploading new found hashes. From that perspective, it was a positive change. However, KL had to maintain a fail2ban system instead. We experienced false positives with this a few times, with some members being locked out for 5 minutes at a time. Since we had so many unique IPs, this wasn't really a problem - but it is something to look into.

The main contest information page initially stated that Tor IPs were already banned, but this statement was later removed, which we found to be curious. Tor did indeed appear to be blocked prior to the start of the contest, but at least some exit nodes were not blocked when we checked during play.

Team organization

For our real-time communication, we used a combination of Google spreadsheets (for brainstorming and state-keeping), our own Hash Management system (List Condense), IRC on our team channel, and a Teamspeak server. Most brainstorming was conducted via IRC.

We also added two new members to the Team:

- Chick3nman (Sam Croley)
- TychoTithonus (Royce Williams)

Software used

- hashcat (v4.0.0-rc1, available at https://hashcat.net/beta/)
- Hashtopussy (v0.4.2, latest from GitHub)
- princeprocessor (v0.21, latest from GitHub)
- maskprocessor (v0.73, latest from GitHub)
- John the Ripper (community edition, latest from GitHub)
- MDXfind (v0.97)
- Team Hashcat LC (List Condense) (private tool)
- Google Docs spreadsheet
- Various common command line tools

Hardware used

- NVIDIA 750Ti (1x)
- NVIDIA 970 (4x)
- NVIDIA 980 (10x)
- NVIDIA 980Ti (6x)
- NVIDIA 1070 (4x)
- NVIDIA 1080 (45x)
- NVIDIA 1080Ti (16x)
- AMD 290 (4x)
- AMD 290X (4x)
- AMD RX480 (1x)
- AMD RX580 (1x)
- AMD Vega64 (1x)

The combinated NTLM (single hash) performance of this setup was around 3400 GH/s. At this cracking rate, it would take only two days to crack all NTLM ASCII passwords up to and including length 9.

The power consumption was 250 kW/h, which would cost about 70 € to run it for the ~15 hours spent on the contest. Not all devices were running all the time, but this estimate provides an upper bound for overall cluster capability.

Software usage and workflow

For most of the time, especially in the first few minutes of each challenge, each member ran their own attacks on their own systems. This is more efficient in the early stages because it allows a variety of diverse analysis to happen quickly.

For some challenges, when the cluster's combined power was needed, we used our team-internal Hashtopussy (HtP) instance. This was the first time we used HtP for a contest and it worked well. We did not encounter many technical problems with HtP, and it ran very robustly. Prior to the contest, we did encounter issues where the HtP client depended on specific output from the hashcat executable that had changed between 3.6.x and the 4.x beta. The HtP maintainers acted very quickly to address this issue and push a fix to GitHub that maintained compatibility with both old and new versions.

While each member could create (in theory) their own attack using HtP tasks, we decided to let only one member control the HtP tasks to avoid conflicts. The use of HtP should provide a nice advantage for contests, because it naturally produces a shared record of which attacks have been carried out so far (though we did not use it for this as much as we should have). Another advantage of HtP is that it enables a smooth way to queue up new attacks while other attacks are running, and then fall back to the next tasks automatically.

For hashcat, which we used for all challenges (except for the PKZip and PGP ones), we simply used the current hashcat beta (v4.0.0-rc1) unmodified as can be downloaded by anyone from https://hashcat.net/beta/. We had no problems during the contest with this version and no changes were needed.

The Challenges

Challenge #1: MD5 (13:01-13:25 UTC, 24m)

This challenge was a straightforward list of MD5s, which provided a good introduction to what the format of the contest would be, as it should be approachable for new and intermediate-level password cracking enthusiasts. The first hint provided by KoreLogic prior to the contest (about case-insensitive replacement of 'slash' with '/') was solid, and should have been helpful for beginners. The use of simple English word lists and basic rules - to change case and perform simple 'leet' substitutions (4 -> a, 0 -> O, etc.) - would get you most of the way there. The other words for punctuation (d0t, dot, D4sh) could be cracked with either the hint, leet rules, or simply from studying the other cracks around them. Combinator or PRINCE attacks using basic English words would help with the couple of strings that were multiple words concatenated together. Some longer strings ('HTTPS://', most.html) should have been reachable once it becomes apparent that an URL is in play. A wordlist drawn from the contest site itself would pick up 'KoreLogic' if it was missing from dictionaries. For us, most manual analysis was not necessary, as most of the words were already represented in most team members' existing wordlists or via common rulesets or pre-scripted attacks. The 'most.html' took us a couple of minutes, but in retrospect should have been easier in context.

Challenge #2: SHA256 (13:25-13:45 UTC, 20m)

Quickly identifying the hash type, basic cracking started immediately, using similar approaches. Combinator, leet, and a wordlist pulled from the contest site itself and from the found passwords from the first challenge ("loopback" work) cover the rest. Repeated slashes should have been found in common wordlists, and finding them provided insight into other ways that URLs might be presented in future challenges.

Challenge #3: NSLDAP / Base64 SHA-1 (13:45-14:02 UTC, 17m)

It took a couple of minutes to confirm hash type because the hash was truncated (the {SHA} signature substring was cut off from the hash). The interesting alternate punctuation word "daught" was found almost immediately (likely via candidate word "daughter" and rule "]]"). Combinator, basic rules, and loopback covered most of the rest.

Challenge #4: Cisco type 4 (SHA256) (14:02-14:23 UTC, 21m)

The hash-type was found in the first 5 minutes. Attacking the plain didn't prove to be any issue due to similar plains and toggle-cases. The last password plaintext left was "ToPingUs", which we narrowed down pretty quickly by deduction from the surrounding plains.

Challenge #5: LM:NTLM (plus pastebin) (14:23-15:22 UTC, 59m)

It took longer to identify the hash type than it should have - almost 20 minutes. Attacking the LM hashes first (case-insensitive, 7 characters max) provided information to inform cracking the NTLM. The last four found passwords were Pastebin IDs. The first three pastes were identical, and the fourth was almost the same, just with the last column of the same hashes appended again.

The transformation of the case-insensitive passwords chunks from LM to the case-sensitive NTLM password was carried out as follows:

1. Brute-force all of the LM hashes (this only takes a few minutes for the entire keyspace using modern GPU)
2. Combine the LM plaintext password chunks (using combinator.bin from hashcat-utils) into a wordlist. This is an inefficient method for larger attacks, but good enough to produce a fast result for the contest
3. Attack the NTLM hashes with the previously generated wordlist, combined with the toggles5.rule ruleset (to try all of the different uPpEr-LOweR case combinations)
4. If not all of the hashes can be cracked this way, use hashcat-legacy in -a 2 mode, because this mode is guaranteed to iterate through all upper-lower case variants

This is the first challenge for which specific contest experience (keeping founds from a previous contest) could theoretically provide an advantage, because some of the found passwords were later discovered to have come from a previous contest.

There was also where some confusion began that carried through the rest of the contest. We treated the set of hashes revealed by the pastes as a separate challenge, which caused our challenge numbering system to diverge from what we later learned to be the public one. It became clear later when KoreLogic referred to specific challenges by number in some tweets that we had diverged. For future challenges of this type, we recommend to KoreLogic that some kind of unique identifier or level indicator be included in each challenge, to eliminate this confusion in future contests.

Challenge #6: sha1(md5(pass)) (15:22-15:37 UTC, 15m)

The hash type was identified in the first couple of minutes. We blew through the plains with no problems whatsoever.

Challenge #7: vBulletin, defective salt (15:37-16:07 UTC, 30m)

The previous challenge gave a good hint about how to deal with these hashes. It was also very clear when we saw that all the vBulletin salts were truncated at length 29, when that format is usually length 30 (the default vBulletin setting). To solve this, we did the following:

1. Use maskprocessor to generate a wordlist containing all 95 possible characters which are allowed in a vBulletin salt
2. Use combinator.bin with the hashlist on the left side and the previous generated wordlist on the right side
3. The result was a hashlist that was 95 times the size of the original

There were no problems with the plains once the salt was fixed, except for the last one ("PASS.sql"), which took a few minutes due to a new file extension that hadn't been seen in any previous challenges.

Challenge #8: sha1($pass.$salt) (16:07-05:03 UTC, 12h56m)

Detecting the missing ":" took us a couple of minutes and was done by extending the hashfile with "-a6 ?a". Once we confirmed the salt was correct, we cracked most hashes fairly quickly again - but hit a roadblock about one hour into the challenge. We solved all but one hash, which was crucial to build to URL since it represented the last directory. The text of the URL was from the popular song "What is Love" by Haddaway:

"what/islove/baby/dont/hurtme/donthurt/MEEEEEEE/ [????] /WriteThat.WAV"

This took us for quite a ride, since the lyrics suggest that the missing password was something based on the phrase "no more". This turned out to just be a red herring. In fact, it took us 13 hours in which we almost tore our hair out waiting for all kinds of attacks to finish (including a brute-force of all valid chars up to 9). We finally found the last plain ("DidBonJovi") using a 3x-right-rotate rule "}}}" applied to an old CMIYC dict.

Challenge #9: MD5 (in WAV file) (05:03-05:34 UTC, 31m)

The .WAV file in the previous level contained the sound of someone trying to guess their own password and mashing the keyboard, but this turned out to be unrelated to the contest. Instead, examining the raw WAV file itself revealed 11 MD5 hashes appended to the end. Almost all of these hashes were keyboard walks. For the missing ones, we also tried to find out the length of password by analysing the .wav file with a WAV editor, and counting the click sounds the keyboard makes when the person hit it, but this proved to be unrelated. A hint from KoreLogic on Twitter indicated that these were directories on the contest server itself. A check for the existence of these directories required carefully not triggering the fail2ban threshold on the servers. We soon discovered that they were each a subdirectory in a single path, in order as presented in the hash list (except the last one, which was the file, not the directory). This was the URL for the next challenge.

Challenge #10: sha1($pass.$salt) (URL) (05:34-06:26 UTC, 52m)

Most of the plains found in this level were wordlist-driven. Once we found some plains that showed URL-escaped content, we added lists of those to base wordlists to continue the cracks. MDXfind also has a '-c' option that automatically expands characters to their encoded form that was useful here. The hashes.org 'junk' lists were also useful as wordlists here, because those lists contain a lot of HTML and URL-escaped chunks - which was exactly what we needed to get an idea of what this challenge was about. The clue suggests converting the result to lower case alpha, which produces the URL to the next challenge.

Challenge #11: md5crypt (quick brown fox) (06:26-07:00, 34m)

In the md5crypt challenge, there are clearly four sections of founds, separated by sequences of underscores that appear to serve as section separators. Those four sections are:

1. The base wordlist, consisting of simple English words from the classic typing-demonstration sentence “The quick brown fox jumped over the lazy dog”
2. A mysterious section with two harder hashes in it. These hashes were not necessary for solving the puzzle. At this writing, we have not cracked them and we’re not aware of anyone who has.
3. Some candidate plains that might be the longest (and one of them was, or at least we used one of them to move to the next level)
4. The URL and instructions on how to assemble the URL to complete the challenge.

Using combinator attacks with the base wordlist from the first section, most remaining plains in the third section were cracked.

However, this didn’t lead us to the real “longest” password, which was still missing at this point. We simply tried the longest password on the server and it didn’t work, so one of the remaining hashes must have been longer. The longest password at this time was of length 15, but with older (pre 4.0.0) versions of hashcat such md5crypt passwords can be cracked only up to length 16. This is a feature to improve the performance of cracking md5crypt hashes. However, this wasn’t the problem, because we were all using the latest beta version of hashcat, which allows cracking passwords of up to length 256. We were not sure why we hadn’t found any passwords longer than 15.

The problem came from something else: We were using princeprocessor (pp) to crack this multi-word combinator passphrase. By default, pp limits its output to strings not longer than 16 to reduce the total number of password candidates that can be produced if the wordlist contains single-byte characters. This explained why we had only cracked passwords of up to length 15 and not longer. This pp restriction can be easily overridden on the command line by using the --pw-max option. When we set it to 256, we almost instantly cracked the longest password “doglazytheoverjumpsfox”.

This was the final level for which the solutions generate a URL to achieve the next level.

Challenge #12: PKZip containing a tarball (07:00-07:08 UTC, 8m)

First, we dumped the file in hex form, and visually verified that the header matched the standard pkzip header. We renamed the file accordingly. We then extracted the hash using zip2john.

For cracking, we could not use hashcat. While hashcat supports multiple different versions of cracking ZIP/7Zip or rar files, this specific (and very old version) is not supported. So we attacked using bleeding-edge john (the GitHub magnumripper “jumbo” version) instead.

The really easy password (Spring16) was found using an existing wordlist.

Challenge #13: PGP private key (07:08-07:15 UTC, 7m)

Inside the password-protected ZIP file were two files: a PGP private key, and a PGP-encrypted text file.

We imported the private key into GPG to check information about the key and check validity. We then extracted the hash using gpg2john.

The hash was then attacked using bleeding-edge john. At startup, john complained about self-testing being broken. We quickly added the “-skip-self-test” option to continue.

The really easy password (Summer17) was found using an existing wordlist.

Finish (07:17 UTC)

The final challenge contained instructions to send specific information to a specific email address. The information included some long random strings. As an independent attestation of approximate finish time, we also posted a hash of one of the random strings to Twitter.

Fails and observations

- We had a lot of hardware available, but it wasn't needed to solve the challenges. The contest could have been done entirely with a single GPU
- All our preparations and special tools developed for older CMIYC-style contests didn't get used (rule-analyser, plain-source-finder…), so that time was wasted
- Progressed too quickly through the challenges to be able to fully coordinate people properly
- If the challenges would have contained more hashes, it would have been a problem assembling the clues/next urls
- As mentioned earlier, we recommend that KoreLogic include some kind of unique identifier in each challenge

Final notes

A single unsolved challenge can cost a team all advantage they build to other teams by solving all previous challenges very fast. Our suggestion would be to have a "get out of jail card". Each team can use this card one time.

The fact that there were no prizes for the winning teams is not that important to us. Winning the title is rewarding enough.

Team hashcat wants to thank KoreLogic for organizing this contest again. The new structure made it much more exciting than the previous contests in which participants had to deal with cracking super slow hashes 90% of the time. Generally, cracking fast hashes makes the contest less boring and has the additional benefit that it can be solved before the contest has officially ended. We loved this contest!