CRACK ME IF YOU CAN

2024 DEBRIEF

Team Hashcat

8 September 2024

Table of Contents

About the Contest

Crack Me If You Can (CMIYC) is an annual password cracking competition created and hosted by KoreLogic. It is most frequently held during the annual DEF CON hacker conference in Las Vegas, NV (with two historical exceptions when CMIYC was held during DerbyCon in Louisville, KY). This competition involves cracking cryptographic password hashes and password-protected files such as documents, archives, and disk images from various artificial sources, with plaintext values typically following a common pattern or theme. Point values for each successful crack typically scale with the difficulty of the password hashing function or underlying KDF. The contest is partitioned into two classes: the cut-throat Pro class, and the more casual Street class.

About the Team

Founded in 2010, Team Hashcat is a static, hand-selected fraternity of professional password crackers who have proven themselves worthy of representing the Hashcat name. Organized and led by Hashcat founder *atom* and managed by team member *Dropdead*, Team Hashcat has taken First Place in fifteen password cracking competitions over the past fourteen years, including ten First Place CMIYC victories. Team Hashcat represents a subset of the best the password cracking community has to offer.

The following 18 team members actively participated in this year's CMIYC competition:

atom	baybedoll	blandyuk	Chick3nman	dropdead
epixoip	kontrast23	kryczek	matrix	m3g9tr0n
N|GHT5	philsmd	rurapenthe	T0XIC	TychoTithonus
unix-ninja	Xanadrel	xmisery		

Organization & Planning

The 15th annual CMIYC competition took place during DEF CON 32 in Las Vegas, NV, running from 11:00 AM PDT on August 9, 2024, to 11:00 AM PDT on August 11, 2024. Team Hashcat, as expected, competed in the Pro class. With team members scattered across the globe — some on-site in Las Vegas and most others working remotely — we utilized Discord for both real-time and asynchronous communication, Google Sheets for strategic planning, and our custom-built collaborative cracking platform, *List Condense* (LC), developed and maintained by very own *Xanadrel*, for operational efficiency.

Software Stack

Team Hashcat has primarily utilized the same software stack for the past 10 years, reflecting the fact that there hasn't been any monumental shift in tooling within the password cracking landscape. However, as with any complex competition, we did have to integrate some "exotic" software tailored to address a few of the more unusual challenges we encountered, such as handling obscure hash types and proprietary algorithms. Despite these additions, the lack of significant advancements in mainstream tooling speaks to the maturity of the current ecosystem. We continue to rely on the established, battle-tested methods that have proven effective, adapting only when confronted with specific edge cases that require unique solutions.

The following software was utilized by Team Hashcat for this competition:

Name	Version	Link
List Condense (LC)	0.1.0b	Internal collaboration platform, not for distribution
hashcat-lc	6.2.6+LC	Internal fork of Hashcat that integrates with the LC API
hashcat-utils	1.9	https://github.com/hashcat/hashcat-utils
John the Ripper	Bleeding Jumbo	https://github.com/openwall/john
PCFG Cracker	4.5	https://github.com/lakiw/pcfg_cracker
princeprocessor	0.22	https://github.com/hashcat/princeprocessor
PACK2	0.1.0	https://github.com/hops/pack2
Elcomsoft AAPR	2.0	https://archive.org/details/aapr20/

Hardware Stack

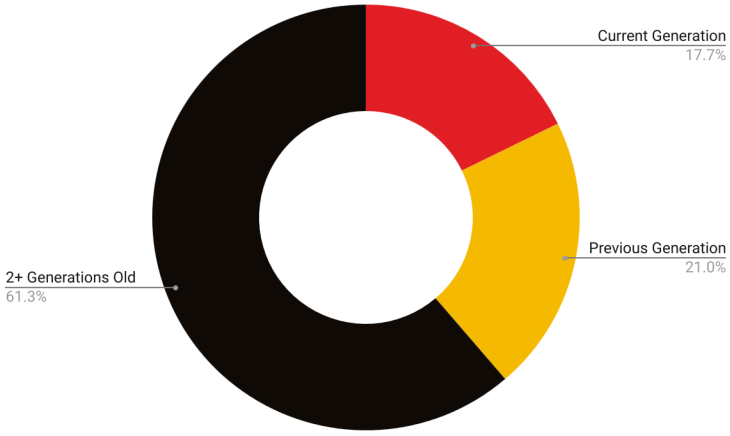
Most members of Team Hashcat utilize personal hardware that is readily accessible to them. Most commonly, these are personal desktop computers with mid- to high-end CPUs and consumer-grade GPUs. Occasionally, cloud assets and special-purpose hardware such as FPGAs are deployed for specific challenges.

Team Hashcat utilized a total of 47 FPGA boards and 62 GPUs for this competition:

Count	Model
47	ZTEX 1.15y FPGA boards (4 x Xilinx Spartan-6 LX150 FPGAs per board)
9	NVIDIA GTX 1080 Ti
8	NVIDIA RTX 4090
8	NVIDIA Tesla V100
7	NVIDIA GTX 2080 Ti
5	NVIDIA GTX 1070 Ti
4	NVIDIA GTX 1080
3	NVIDIA RTX A2000
3	NVIDIA RTX 3080 Ti
3	NVIDIA RTX 3060 Ti
3	NVIDIA GTX 980 Ti
2	NVIDIA Titan Xp
2	AMD Radeon RX 6900 XT
2	NVIDIA RTX 3090
1	NVIDIA RTX 4080 Ti
1	NVIDIA RTX 4080
1	AMD Radeon RX 7900 XTX

This year, Team Hashcat utilized 21% fewer GPU resources compared to last year, marking a 51% reduction from CMIYC 2021 and highlighting our commitment to reducing our carbon footprint in accordance with global climate initiatives. We also did not utilize any cloud resources this year, instead relying entirely on personally owned hardware for this contest. And, as expected, our ZTEX FPGA boards were dedicated exclusively to cracking bcrypt hashes.

Similar to last year, fewer than 20% of our GPUs were current-generation models. More than 60% of our GPUs were two or more generations old, with the three generations old GTX 1080 Ti remaining the most represented GPU amongst team members. This reinforces the notion that cutting-edge hardware isn't essential for success, and continues to underscore that older hardware is not a barrier to competitiveness.



Distribution of graphics processors by age.

A perpetual theme with every contest, we once again struggled to keep our hardware 100% utilized, as the very nature of this competition leads to most time being spent on pattern analysis and attack preparation.

Competition Narrative

KoreLogic provided the following synopsis for this year's competition:

```
Zoogleta has been scheming to corporatize and enshittify the Internet through regulatory capture,
squashing indy devs, and commodifying users.

You've been contacted by journalists and whistleblowers who need help sifting through some big
dumps of encrypted data and password hashes.

Help them so they can publish the smoking gun, crash Zoogleta's stock price, and get their
leadership and the corrupt politicians they own arrested by exposing their internal dirt, for
great justice.

Time is of the essence! You will have 48 hours to crack as many files and hashes as possible.
```

As tradition, each team member independently downloaded and decrypted the GPG-encrypted files dropped by KoreLogic at the start of the contest. This year, we were greeted with two files — one a text file containing 30,685 hashes, and the other a tarball. Naturally, we focused our efforts on the hash list first, segmenting and analyzing it based on the apparent formats.

The hashes came prefixed with usernames, immediately signaling that this was some sort of database dump scenario, likely tied to that notorious "Zoogleta" company. It wasn't immediately clear whether these were employee accounts or some other group of users, but the contest description made one thing certain: we were tasked with digging through these dumps to find something damning. Game on—let's see what this dataset has to offer.

The hash list was a mixed bag. Some of the formats were natively supported by Hashcat, so we could dive straight in:

- ➔ NTLM (mode 1000)
- ➔ Raw SHA-1 (mode 100)
- ➔ bcrypt (mode 3200)
- ➔ MS-AzureSync PBKDF2-HMAC-SHA256 (mode 12800)
- ➔ RAdmin (mode 29200)

But as always, there were plenty of edge cases that required additional legwork. A few of the hashes we recognized but did not support in Hashcat, and others didn't immediately match any standard algorithms.

- ➔ Variable length hex-encoded strings that we identified as the "striphashes"
- ➔ RC2
- ➔ sm3crypt
- ➔ Apache Shiro (Argon2id)
- ➔ Something with an `x-isSHA512` prefix, which we identified as SAPSHA512
- ➔ An unknown bcrypt variant with a `$2k$12$` prefix, which we identified as bkr256
- ➔ Another unknown bcrypt variant with a `$2b$12$` prefix, which we identified as bcr256

We initially did not know the points values for each of these algorithms, but it quickly became obvious that the majority would come from the bcrypt variants, which put us on edge right away. KoreLogic has been ramping up the focus on bcrypt over the last two contests, and this year 84% of the points came from bcrypt

variants. We get that bcrypt adoption is finally reaching mainstream status, largely due to it being the default in PHP 7+ and several popular webapp frameworks, and that's reflected in the growing number of bcrypt hashes in public leaks. But most modern environments have moved on, so why such a heavy focus on it? If their goal is to push us to optimize the bcrypt kernel further, they'll be disappointed — we've already wrung every last drop of optimization possible out of our bcrypt kernel. If KoreLogic truly wanted to encourage skillful pattern recognition and exploitation while leveling the playing field, they'd be better off using PBKDF2 with a high iteration count, which is more GPU-friendly and doesn't require FPGAs. New teams, take note.

All that aside, we didn't waste any time. Everything was immediately funneled into List Condense, but we also set up a Google Sheet to organize the user:hash pairs and prepare for correlation with external data. This structure would later prove indispensable. But an hour into the contest, reality set in: KoreLogic didn't just throw one curveball — they threw several. We only had four lists ready for immediate cracking. The rest were still unidentified, unverified, or just out of reach since Hashcat didn't natively support them. It was a scramble to get the remaining hashes prepped.

Initially, the striphashes looked like truncated versions of SHA1 hashes. We quickly cracked some of the 39-character variants using `mdxfind`, but had no luck with the shorter ones when treating them as simple truncations. At first, we explored a "character-driven" approach, assuming hex characters had been randomly removed from any location within the hash string. This led us to generating a full hash list by inserting all possible missing characters at any position to restore the hashes to their original length. While this method worked for the 38-character hashes, the sheer volume of combinations for shorter ones made it clear that scaling this approach would be inefficient. Then, *Xanadrel* made a breakthrough discovery — he realized the missing characters were always zeroes. Armed with this knowledge, we were able to restore the hashes by padding them with zeroes in every possible position until we reached the full 40-character length.

Turning towards the file dumps, the first dump contained 201 ARJ-compressed archives — a format that dates back to 1991 and was once a leading compression tool for MS-DOS and Windows 3.x. However, cracking ARJ archives proved challenging due to the lack of modern tools specifically designed for this task. We decided to approach the problem using multiple methods. Our initial attempt was through a multithreaded Python implementation, leveraging "patoolib" to extract the archives while testing password validity. Then, we turned to legacy tools like "Elcomsoft Advanced ARJ Password Recovery" (AAPR) and "Yet Another ARJ Cracker" (YAAC), both relics from the early 2000s. These tools required dedicated systems to run, with Elcomsoft's ARJ cracker only functioning in Windows 95 compatibility mode. Despite the outdated nature of the software, it was necessary to resort to these ancient utilities to tackle this format.

We found RAdmin3 hashes buried in a file called `radmin.reg`, which appeared to be a registry export from Windows. While we had our `radmin3_to_hashcat.pl` script for extracting RAdmin3 hashes, it only handled one hash at a time, which was far from practical given the volume we were dealing with. We ended up having to write a custom script to pull out all the individual hashes at once. Though it sounds straightforward, this turned out to be more tedious than expected. Ultimately, it came down to some regex trickery in EditPlus to get everything extracted cleanly. What should have been simple took longer than anticipated, but in the end, we got it done.

In the second file dump, we discovered the "gen_rc2.py" script, which contained the exact algorithm used to generate the RC2 hashes. Using this information, we developed and implemented an optimized version of the RC2 cracking process in a custom multiprocessing Python script. This approach allowed us to efficiently distribute the workload across multiple cores, significantly speeding up the attack and ultimately leading to the successful cracking of the RC2 hashes.

By the time the first night hit (for the Europeans, anyway), *atom* had already contributed solutions to crack the Apache Shiro and SAPSHA512 (mode 35000) hashes. For Apache Shiro, it involved tweaking the settings to make the hashes compatible with John the Ripper. SAPSHA512 required more work: a new plugin had to be written for Hashcat, leveraging the existing SAPSHA256 code as a foundation. During the second day, *atom* also created an implementation of sm3crypt (new hash-mode 35100). Both new implementations will be released to the public in the upcoming hashcat release. Having an sm3crypt module really boosted our progress on those algos, though it didn't save us from plateauing later that day. Our progress on the high value algorithms was still pretty slow.

By this point in the contest, we had solidified our strategy:

- ➔ Target the fast algorithms first (NTLM, SHA1, RAdmin3)
- ➔ Apply any successful patterns to the slower algorithms (Argon2id, sm3crypt, SAPSHA512)
- ➔ Once we were confident in an attack, we would run it against bcrypt

Some of the patterns and themes we identified during the contest were:

- ➔ Indian given names with "YY", "@YY", "YYYY", or "@YYYY" appended
  Akshay12, *Arjun@17*, *Akshay2308*, *Dharshan@2017*
- ➔ Names of Star Trek actors
  Liam Bilby, *Sarah Sisko*, *John Torres*, *Kate Bowman*
- ➔ Star Trek characters, titles, and ranks, particularly in "<adverb/rank> <noun/surname>" format
  Admiral Cartwright, Alien Commander, *Bridge Officer*, *Cardassian Prisoner*
- ➔ Phrases from books, sometimes with space replaced with another special character
  You were right. David reached the top of Arasaka Tower.
  His friendship was an inspiration, his love a blessing.
  Here I lie. I wanted it all, but I lost everything.
  He@drops his battered
  He takes the#cigarette
- ➔ QWERTY to Dvorak mappings
  Dapc@2014, *Cbeca2012*, *Ipcuucb08*, *Dppcedg23*
- ➔ Cybersecurity/hacktivism
  Hack4Justice, *CyberwarriOr99!*, *CtrlAltWin8**
- ➔ Minga vanity strings
  minga1mingaz!, *minganbing2024*, *mingalingbung0$*, *MINGAGUA1U8@*
- ➔ Pop culture references
  metallica!, *spongebob!*, *livingonapraye11*, *blink_182*
- ➔ Financial/banking
  Citi1bank, *InvestmentBanking#*, *Citi1bank*, *CorporateFinance*

- ➔ IT terminology
  Network#application, *Ethernet$speeds*, *Processor+speeds*
- ➔ Fantasy/RPG
  Paladin#1, *Mage2021*, *DragonSlayer!*
- ➔ Date Formats
  25Nov2006, *2009-08-20*, *08-09-2024*

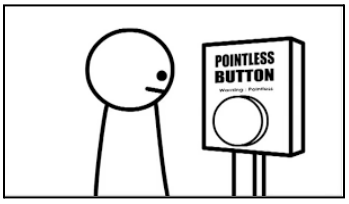
Additionally, *epixoip* leveraged Lakiw's PCFG (Probabilistic Context-Free Grammar) Cracker to great effect by training a custom PCFG model on all of our cracked plaintexts, which enabled us to generate new candidate passwords with structures that closely mirrored the patterns we had already observed in our successful cracks. This had the greatest effect against the Argon2id and SAPSHA512 hashes.

To streamline our efforts and track correlations between users, hashes, and plaintexts, Dropdead set up a Google Sheet that mapped all hashes to their associated data, including:

- ➔ Username
- ➔ Plaintext
- ➔ Algorithm
- ➔ Original and formatted hashes
- ➔ Tags
- ➔ Department
- ➔ Cracks from files and hints
- ➔ Street-level cracks based on usernames
- ➔ Dvorak-to-QWERTY mappings

This system helped us spot potential correlations, but we often found ourselves chasing dead ends, searching for patterns that simply didn't exist — much like real-world cracking.

And then there were the .zip files... It was clear from the beginning that these would contain valuable information. However, this .zip file came with a twist: each file within the archive had its own password. KoreLogic intended for users to discover this feature, where individual files in the same .zip can have different passwords. Unfortunately, we initially picked the only uncrackable file, which led to a lot of wasted effort. In hindsight, we realize this wasn't a matter of luck. We should have recognized that mixed-password zips could contain uncrackable files, but after testing a few individual hashes without success, we deprioritized the zips and moved on to other tasks. The lesson here is that persistence and a methodical approach might have revealed the other crackable files sooner, instead of abandoning the effort too early.



Additionally, as in previous years, we attempted to crack the Street class hashes to identify overlapping plaintexts and patterns. Unsurprisingly, we saw the same familiar patterns and sources emerge, which reaffirmed what we already knew, but unfortunately without yielding any new patterns or sources. Given the high cost of fully pursuing these patterns, we decided against dedicating more resources to them. We also took a shot at the Street .zip files, hoping for better luck, but we ran into the same frustrating issue: we ended up selecting the one hash that was uncrackable. This reinforced our earlier decision to not invest further time in exploring the Street class, as the payoff didn't justify the effort.

And finally, the timecodes. We first encountered the timecodes after KoreLogic released one of their hints, but figuring out how to crack the expensive bcrypt hashes tied to these timecodes took some time. *Chick3nman* started by cracking the base timecodes from the NTLM hash set using a simple mask and combinator attack. These base cracks led to unlocking user .zip files from an earlier file drop, each containing new timecode information along with a source. The first breakthrough came when Chick3nman very rapidly realized that the random-looking string next to the word “transcript” was a YouTube video key, and that the timestamps were pulled directly from YouTube transcripts. This discovery proved pivotal. From there, *epixoip* wrote a script that, when provided with a YouTube video key, used the YouTube API to download the video’s transcript, extract the relevant lines based on the timestamps, and pull the exact words needed to construct the password. This script allowed us to crack hundreds of bcrypt hashes at an impressive pace, with *Chick3nman* doing the bulk of the work using *epixoip’s* automated process.

Meanwhile, *atom* took on the task of parsing books, which was a much more difficult challenge. Several team members helped track down copies of the books, some of which had to be cleaned up manually before *atom* could use them in his cracking efforts. This was near the end of the contest, and *Dropdead’s* spreadsheet work played a crucial role in tying all the data together. However, automating the book-based timecodes was a major headache. YouTube transcripts were relatively straightforward: timestamped in the hour:minute:second format, with each word simply counted from the start of the relevant section. The books, on the other hand, followed an imprecise structure (chapter:paragraph:word? page:line:word? how could we know?), and gathering the correct words from various fanfictions and books felt like a nearly impossible task. KoreLogic eventually revealed that they had used OCR-scanned copies of physical books, which was strange given the online nature of the contest. We had naturally reached for digital copies like .epub or .pdf, but how could we even be sure we had the same versions KoreLogic used? The whole process for the books felt poorly thought out and unnecessarily frustrating.

Also, a major shout-out to *RuraPenthe* for throwing a ton of points on the board in the final hour. His expansive Star Trek knowledge really came in clutch and he cracked an insane amount of Argon2id hashes in a very short amount of time.

In Conclusion

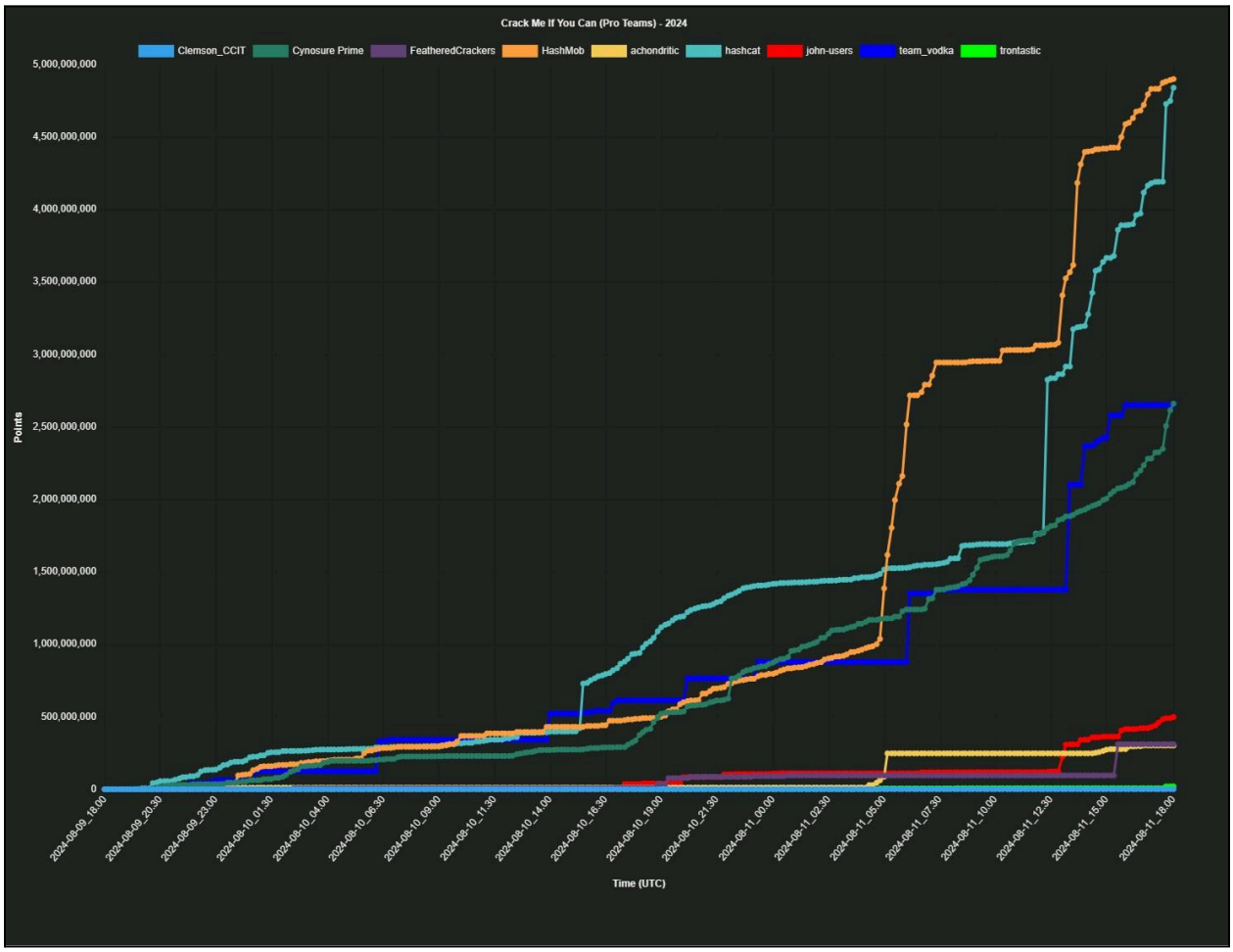


Photo finish, CMIYC edition.

This year's Crack Me If You Can contest centered around a fictional scenario where teams were tasked with helping journalists and whistleblowers expose an evil corporation, Zoogleta. While the concept was novel and added an interesting narrative, it lacked the immersive connection we've felt in previous contests and we felt wholly detached from the plot. In the end, it didn't matter who the whistleblower was, did it? The challenges were intense — borderline undoable within the given timeframe — but in a way, that's exactly what pushes teams like ours to evolve. With around 120 billion points on the table and the winning team only managing to pull in around 5 billion, it raises the question: Is KoreLogic encouraging teams to step up, or is this the expected outcome?

One of the most significant pain points was the constant need to search for, validate, and map sources into actionable attacks. While this added a layer of complexity, it also highlighted an area where more sophisticated tooling could make a difference, though such tools might only be useful in the context of this contest.

The biggest takeaway for us this year was that we've reached an inflection point. For the first time, the reduction in hardware became a limiting factor despite still not fully utilizing all of the resources we had, forcing us to rethink how we allocate resources. Beyond that, the most pressing challenge remains how to better correlate data points more efficiently. KoreLogic has continued to push for innovation and efficiency in this area, aligning with their mission to simulate real-world cracking challenges. Overall, great work from KoreLogic! The contest has clearly evolved, and so must the balance between strategy, tooling, and hardware.

We want to extend our thanks to all the other Pro teams — HashMob, Cynosure Prime, team vodka, achondritic, and john-users — for the fierce competition. It was great to see so many new teams join the fray, as more competition makes for a better contest all around. And congrats to HashMob on their first-ever CMIYC victory – something we'll ensure never happens again ;)

Until next time,
~ Team Hashcat

Prep for next CMIYC.

Please direct any questions or comments to the Hashcat Discord at https://hashcat.net/discord