Hashcat 7.1.0 - Release Notes

This is a minor release, but an important one. It comes just two weeks after the major v7.0.0 update, as part of our effort to keep release cycles shorter than in the past.

Although two weeks may seem quick, this version includes several important bug fixes along with notable new features and hash modes, which makes a v7.1.0 release fully justified.

---

1. Algorithms

1.1. KeePass (KDBX v4)

This is likely the most prominent new hash type in this release. While KeePass (KDBX v2/v3) was already supported, KeePass (KDBX v4) introduced Argon2 support, which kept it out of reach for some time. Since the Argon2 KDF has been fully supported since hashcat v7.0.0, adding KDBX v4 was the next logical step, as we could reuse the post-KDF code from KDBX v2/v3.

1.2. LUKS2 (Argon2i KDF type)

While we expected LUKS2 containers or partitions in the wild to use Argon2id or PBKDF2, we have also found cases using Argon2i. Our Argon2 kernel already supports all Argon2 variants, so adding Argon2i was simply a matter of enabling it in the module. The extraction tool already supported this variant.

1.3. Blockchain, My Wallet, Legacy Wallets

This new hash mode supports loading very early Blockchain wallets that used only a single iteration of PBKDF2 combined with AES in OFB mode. Because it uses only a single iteration, we replaced the PBKDF2 call with pure HMAC operations and switched to a fast hash candidate generator in a nested loop, resulting in a significant performance boost, thus showing it here (RTX 4090):

```
Speed.#01........:  5516.8 MH/s
```

As a side note, these wallets are typically entirely base64 encoded.

1.4. MD6

In preparation for the Jabbercracky Password Cracking Contest at DEF CON 33, we identified a potential need for MD6-256 support. While we have no confirmed evidence that this algorithm is currently used in practical applications, we developed and integrated it to ensure readiness in case it becomes relevant. At present, only the optimized implementation is included, which supports password lengths up to 32 characters. This could be extended in the future by implementing a true pure-kernel library version, allowing much longer password lengths..

2. Improvements

2.1. Attack-Modes: Use 64-bit counters for amplifier keyspace

Hashcat now uses 64-bit counters for the amplifier keyspace (previously limited to 32-bit), allowing greater flexibility in attack design. All attack-modes are supported, but the primary impact is on the combinator and both hybrid attack-modes. This change could be especially relevant if you have ever seen the following message in previous hashcat runs:

```
Integer overflow detected in ...
```

Note there is still a 64 bit restriction for the product of the base word count multiplied by the amplifier count, but you can now distribute them freely and gain more flexibility. For example, if your base word count requires 40 bits, your amplifier count can only be 24 bits, or vice versa, just as it has always been in pure kernel mode.

2.2. Host Memory: Update method to query free host memory

Previously we used sysinfo(), but when large amounts of memory are used by the filesystem cache, which still counts as potentially free, this method fails. Depending on your system configuration, this can consume a large amount of memory when files are cached in host memory, which can cause hashcat to report this incorrect error:

```
Not enough allocatable device memory or free host memory ...
```

This was not only incorrect but also often misunderstood as the compute device (GPU) being out of memory, when in reality it was the host memory that was insufficient. We now query /proc/meminfo, if it exists, and check the MemAvailable attribute. Only if the file is missing do we fall back to sysinfo().

2.3. Docker: Add initial support for running hashcat inside Docker

To improve ease of use, we now provide a container template for packaging and running hashcat. This setup includes optional third-party tools like pyenv to help you quickly try out the new Python Bridge feature without additional system setup.

Due to the fundamental differences between CUDA, HIP, and OpenCL, a separate container is required for each compute backend. As a starting point, we provide support for CUDA with Ubuntu 24.04 as the base image.

Users can choose to build hashcat from source or use the official release binaries. GPU passthrough must be enabled in your Docker setup. For full instructions, see `RUNTIME_Docker.md`.

```
$ docker build -f docker/runtime.cuda.ubuntu24.release -t hashcat .
$ docker run --rm --gpus=all -it hashcat bash
root@d1d5c5b61432:~/hashcat# ./hashcat.bin -I
hashcat (v7.1.0) starting in backend information mode

CUDA Info:
==========

CUDA.Version.: 12.9

Backend Device ID #01
  Name...........: NVIDIA GeForce RTX 4090
  Processor(s)...: 128
  Preferred.Thrd.: 32
  Clock..........: 2565
  Memory.Total...: 24080 MB
  Memory.Free....: 23664 MB
  Memory.Unified.: 0
  Local.Memory...: 99 KB
  PCI.Addr.BDFe..: 0000:01:00.0
```

2.4. Device Memory: Warn instead of waiting on high GPU memory usage

For algorithms using the module callback module_extra_tuningdb_block(), such as Argon2, KeePass (KDBX 4), LUKS2 or scrypt we changed the behavior when a compute device shows high memory usage. This typically comes from the desktop window manager, desktop applications, or other GPGPU programs using GPU acceleration and occupying memory.

```
Memory usage is too high: 3942866432/25250627584, waiting...
```

In v7.0.0, hashcat printed a warning 10 times with delays between each, allowing the user to close applications and free memory. On systems with multiple GPUs, this could lead to significant wait times if memory was not freed.

We now show a single warning instead of waiting. The user is informed about the expected performance loss in percent. To avoid spamming and false positives, the warning is shown only if memory usage exceeds a default threshold of 15 percent.

2.5. Backends: Enhanced AMD Windows OpenCL and HIP compatibility for legacy models

1. Addressed missing built-ins like __builtin_amdgcn_perm() in the AMD **Windows** OpenCL compiler by replacing AMD specific branching with a generic fallback path. Also updated the HIP dynloader on Windows to try hipGetDevicePropertiesR0600() first and fall back to hipGetDeviceProperties() if needed.

2. Added a workaround for a bug in the AMD **Windows** OpenCL compiler that could cause a crash during JIT compilation on older GCN4 and GCN5 GPUs. The issue was triggered by complex switch() based logic in the UTF-8 to UTF-16 conversion code. Other runtimes such as CUDA and HIP are unaffected.

2.6. Rules: Add early syntax check for solo rules

This change adds an early syntax check for -j and -k rules. If an invalid rule is detected, hashcat now prints a clear error message and stops immediately, avoiding the misleading error:

```
example.dict: empty file
```

Example of new behavior:

```
Invalid or unsupported rule specified ...
```

This issue often arises when users copy hashcat examples from tutorials written for different shells. For example, a tutorial might include -j ^a, which is valid in bash. But in cmd.exe on Windows, the caret character (^) is treated as an escape, so -j ^a becomes -j a before hashcat sees it.

This leads to an invalid rule being applied to every word in the wordlist, which causes internal syntax errors.

Previously these errors were not shown, resulting in an empty wordlist without explanation. Hashcat now reports these rule parsing errors clearly, making it easier to debug solo-rule related issues.

2.7. Device Memory: Do not disable hardware-monitor interface by default in speed-only and progress-only mode

Accurate free memory information is critical for memory-intensive hash modes to perform optimally. This data is only available through low-level APIs such as NVML or sysfs, since CUDA, HIP, and Metal often report delayed or incorrect values, and OpenCL provides no free-memory query at all.

Previously, these low-level queries were blocked by hashcat's default behavior of disabling the hardware-monitor interface on startup, leading to a conflict between features.

This conflict has now been resolved.

2.8. JSON output: Refactor Bridge unit reporting for clarity

Previously, all machine readable and JSON outputs listed each bridge unit individually. This was intended to help machine parsing but caused more confusion than benefit.

The output for status, benchmark, speed-only, and progress-only modes has been rewritten. All output formats, including standard human readable, now present bridge units as a **single** merged device.

This should simplify handling in third party applications.

2.9. Backends: Ignore devices from Microsoft OpenCL D3D12 platform

This OpenCL platform is sometimes installed by users who encounter problems when trying to enable OpenCL support for their hardware.

The Microsoft Store offers a package called "OpenCL and OpenGL Compatibility Pack", and when installed, it adds a new OpenCL platform. The purpose of this platform is not to provide access to dedicated hardware only available through this interface, but rather to create a virtual mapping layer for the OpenCL API on top of Direct3D 12. It enumerates D3D12 adapters (from DXGI) and, for each adapter, creates an OpenCL device that wraps the D3D12 device.

These devices cause more problems than they solve, because they duplicate each OpenCL device we already use in an optimized way. Most users who do not pay attention will not notice this and may suffer a significant performance loss as a result. The best way to prevent this problem for us is to automatically disable all devices reported by this platform.

2.10. Building: Add MAINTAINER_MODE flag to disable hardcoded CPU optimization flags

This flag allows building hashcat without using hardcoded compiler flags like -march=native and -mavx2.

Compile hashcat with `make MAINTAINER_MODE=1` instead of just `make`.

If additional changes are needed to make packaging easier, let us know and we can add them.

2.11. Bash: Add missing parameters to bash completion script

The following options were added to the completion list: --advice-disable --benchmark-max, --benchmark-min --bypass-delay, --bypass-threshold --metal-compiler-runtime --total-candidates --color-cracked -5, -6, -7, -8, --custom-charset5 to --custom-charset8 -v

Just run `extra/tab_completion/install` to update and enjoy tab completion of your hashcat parameters.

2.12. Dependencies: Downgraded unrar source from 6.2.7 to 6.0.5

When we updated all the third party dependencies for hashcat v7.0.0 release we also updated to a more recent unrar sources. This created a lot of problems, because there is some sort of change in the threading model of unrar that is incompatible with our own threading, with the result that kernel hooks which call unrar's unpack functions run into deadlocks, stalling entire hashcat process. Since we need these sources only for unpacking unrar3 packages, it is ok to downgrade these package.

2.13. Benchmark: Update default hash-mode selection

Update the default selection used when a user runs hashcat with the -b option but without the --benchmark-all parameter. This list is based on a handpicked set of hash modes.

Also update the default list used by `tools/benchmark_deep.pl` to include all supported hashcat modes up to this version, with some exceptions. These exceptions are kernels that scale linearly compared to other included modes. Therefore, they do not provide additional insight to the user. Additionally, update some hash mode categorizations to ensure they are sorted into the correct benchmark subsections automatically. Also shorten several oversized hash names to prevent column overflow in the table view.

2.14. Libraries: DES and Blowfish restructure

We have refactored and consolidated shared code for the Blowfish and DES ciphers into the files `inc_cipher_blowfish.c/.h` and `inc_cipher_des.c/.h`.

These cipher implementations can now be reused by plugin developers when adding new algorithms that depend on Blowfish or DES.

2.15. Building: Add initial support for OpenBSD and DragonflyBSD

To improve compatibility with the various BSD-based operating systems, we have extended build support to include OpenBSD, DragonflyBSD, FreeBSD, and NetBSD. Of these, OpenBSD and DragonflyBSD had not been supported previously.

It is still unclear which compute runtimes are available for these platforms, particularly for GPU acceleration. For pure CPU workloads, the pocl OpenCL runtime may work on them, although this has not yet been tested.

2.16. Other changes

- Argon2: Add argon2_init() wrappers to support private address space
- Change hash-category for hash-modes 25600, 25800, 28400 and 30600 to HASH_CATEGORY_RAW_HASH_SALTED
- Suppress optimization advice in final approach mode
- Backend: Split backend_session_begin() into smaller compute-runtime specific functions
- Sanity check: Abort if a custom-charset is defined on the command line and a mask file is used
- Building: Disable Argon2 optimized path on RISC-V
- Shared: improved 32-bit and 64-bit add and multiply overflow helper functions

3. Python Bridge

As already mentioned above, team hashcat participated in the Jabbercracky Password Cracking Contest at DEF CON 33. This event provided an ideal opportunity to test how effectively the Python Bridge could be used in a CTF scenario.

We have resolved all identified issues and applied the lessons learned through the following fixes:

- Fix unsalted hashlist support

The Python bridge was originally designed for salted and slow hashes. For this reason, it was implemented as an outside mode, meaning the password candidate generator runs outside the deepest guessing nested loop. In outside mode, a special flag is required to allow multi-hashing when no salt is involved or when same-salt needs to be supported, and that flag was not enabled by default. This has been fixed.

- Fix the esalt structure, it was too large

For most use cases, the esalt structure size was irrelevant, but in scenarios with 200,000 hashes to crack, the 128 KiB buffer per hash became a multiplier, resulting in a 24 GiB memory requirement just for the esalt data. This was overdesigned because the original use case was too specific. The intent was that developers might handle large buffers, such as encrypted blobs, and need to store all necessary data for decryption in Python code. The buffer size has now been reduced to 2 KiB, but if you need support for larger data blobs in your plugin, you can increase the size in the esalt structure.

- Improve support from 1:1 password-to-hash to 1:N password-to-hashes

The original workflow produced one hash per password, with the expectation that the developer would return either a single test result (for a successful decrypt) or the hash in its final form, creating a 1:1 mapping.

This has been changed because, when using raw primitives, the performance impact is negligible and can even help offset some of the PCIe bottleneck overhead. We now reserve space for 32 return values, most likely hashes.

The Python bridge endpoint will first check the type of the return value. If it is a string, it will be copied into the first element to preserve backward compatibility. If it is a list, the bridge will copy all elements for final processing on the GPU.

- Improve stand-alone debugging of Python Bridge stubs

We improved the framework to make it easier to implement your Python Bridge stub directly, without running hashcat, which allows you to attach a debugger and step through your Python code more naturally.

```
$ echo hashcat | python3 Python/generic_hash_mp.py
33522b0fd9812aa68586f66dba7c17a8ce64344137f9c7d8b11f32a6921c22de
```

The Python Bridge framework takes care of most setup details and includes supporting libraries like `hcshared.py` so you can also step through them.

- Improve salt/esalt debugging of Python Bridge stubs

We have added code to make it easier to inspect how salts and esalts are handled. To kickstart your debugging the framework can export the context object from hashcat and load it into the Python environment, giving you full context aware insight into the cracking process.

- Enable potfile output by default for `-m 73000` and `-m 72000`

Cracked passwords are now saved to the potfile as with any other hash-mode. Potfile support had been disabled during development, but we forgot to re-enable it for the release.

- Contest writeup

We have written a short documentation describing how the Python Bridge was used in the contest after all the fixes were applied. You can read it in our forum post here: https://hashcat.net/forum/thread-13346.html

4. Bugfixes

4.1. Fix broken JSON formatting when using the --status-json flag

There was an extra closing bracket `}` where a comma `,` should have been in the buslanes attribute.

4.2. Fix issue where -k solo-rules were ignored when used with -a 1 and -S

That flag was simply ignored completely. Now it works as expected.

5. List of contributors

We would like to thank the following people who contributed to hashcat v7.1.0:

- AmmoniaAvenue
- botantony
- Chick3nman
- dropdeadfu
- ecostin
- fse-a
- jsteube
- matrix
- NuHarborMartin
- PenguinKeeper7
- philsmd
- thatux
- Xeonacid

These are the GitHub account names, you can find their profile page on:
https://github.com/$ACCOUNTNAME