

Core dump analysis for CVE-2025-20260

Generate the 1GB PDF

```
$ python3 clamshank.py
```

The Python script generates a PDF file named `clam-cve.pdf` intended to exploit CVE-2025-20260, a critical vulnerability in ClamAV. It uses a modified PDF template with a malformed header containing an extra number after the `/Length` field and includes a large (1GB) stream of `ASCII85-encoded` 'A' characters, designed to trigger the vulnerability in the ClamAV parser.

```
(aidynskullz@NoxEternis) - [~/cmu_courses/HOS/CVE1]
$ python3 clamshank.py

(aidynskullz@NoxEternis) - [~/cmu_courses/HOS/CVE1]
$ du -sh clam-cve.pdf
1.1G  clam-cve.pdf
```

Run ClamAV on the malicious PDF

```
$ docker run -it --rm --name clamav-crash \
  --ulimit core=-1 \
  --privileged \
  -v /home/aidynskullz/cmu_courses/HOS/CVE1/clamav-dumps:/dumps \
  --mount
type=bind,source=/home/aidynskullz/cmu_courses/HOS/CVE1,target=/root \
clamav/clamav:1.0.8 \
sh -c "sysctl -w kernel.core_pattern=/dumps/core.%e.%p && clamscan
--max-filesize=2048M --max-scansize=2048M --verbose /root/clam-cve.pdf"
```

Breakdown of the Docker command:

- **Runs a Docker container:** Uses the `clamav/clamav:1.0.8` image.
- **Interactive & Temporary:** The container runs interactively (`-it`) and removes itself when exited (`--rm`). It's named `clamav-crash`.
- **Enables Core Dumps:** Allows the container to generate core dump files (`--ulimit core=-1`) and gives it necessary permissions (`--privileged`).
- **Mounts Exploit File:** Makes the host directory `/home/aidynskullz/cmu_courses/HOS/CVE1` accessible inside the container at `/root`. This is where `clam-cve.pdf` resides.

- **Mounts Dump Directory:** Makes the host directory `/home/aidynskullz/cmu_courses/HOS/CVE1/clamav-dumps` accessible inside the container at `/dumps`.
- **Executes Commands:** Runs a shell (`sh -c`) inside the container to:
 - Set the core dump save location to `/dumps`.
 - Run `clamscan` on `/root/clam-cve.pdf` with large file size limits.
- **Generation of Core Dumps:** The core dumps are stored in the `clamav-dumps` directory

```
(aidynskullz@NoxEternis)-[~/cmu_courses/HOS/CVE1]
$ docker run -it --rm --name clamav-crash \
  --ulimit core=-1 \
  --privileged \
  -v /home/aidynskullz/cmu_courses/HOS/CVE1/clamav-dumps:/dumps \
  --mount type=bind,source=/home/aidynskullz/cmu_courses/HOS/CVE1,target=/root \
  clamav/clamav:1.0.8 \
  sh -c "sysctl -w kernel.core_pattern=/dumps/core.%e.%p && clamscan --max-filesize=2048M \
  --max-scansize=2048M --verbose /root/clam-cve.pdf"

kernel.core_pattern = /dumps/core.%e.%p
LibClamAV Warning: *****
LibClamAV Warning: *** The virus database is older than 7 days! ***
LibClamAV Warning: *** Please update it as soon as possible. ***
LibClamAV Warning: *****
Loading: 13s, ETA: 0s [=====>] 8.71M/8.71M sigs
Compiling: 3s, ETA: 0s [=====>] 41/41 tasks

Scanning /root/clam-cve.pdf

(aidynskullz@NoxEternis)-[~/cmu_courses/HOS/CVE1]
$ ls clamav-dumps
core.clamscan.7
```

Start a debug container

```
$ docker run -it --rm --name clamav-debug \
  -v /home/aidynskullz/cmu_courses/HOS/CVE1/clamav-dumps:/dumps \
  --entrypoint /bin/sh \
  clamav/clamav:1.0.8
```

Breakdown of the Docker command:

- **Runs a Docker container:** Uses the `clamav/clamav:1.0.8` image.
- **Interactive & Temporary:** The container runs interactively (`-it`) and removes itself when exited (`--rm`). It's named `clamav-debug`.

- **Mounts Dump Directory:** Makes the host directory `/home/aidynskullz/cmu_courses/H0S/CVE1/clamav-dumps` accessible inside the container at `/dumps` (presumably to access a core dump file).
- **Overrides Entrypoint:** Starts an interactive shell (`/bin/sh`) inside the container instead of the image's default command.

```
(aidynskullz@NoxEternis)-[~/cmu_courses/H0S/CVE1]
$ docker run -it --rm --name clamav-debug \
  -v /home/aidynskullz/cmu_courses/H0S/CVE1/clamav-dumps:/dumps \
  --entrypoint /bin/sh \
  clamav/clamav:1.0.8
/ #
```

Install GDB inside debug container

```
$ apk update
$ apk add gdb
```

GDB is needed for core dump analysis to see exactly what is going on

```
/ # apk update
fetch https://dl-cdn.alpinelinux.org/alpine/v3.22/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.22/community/x86_64/APKINDEX.tar.gz
v3.22.2-85-ge92a8d17c70 [https://dl-cdn.alpinelinux.org/alpine/v3.22/main]
v3.22.2-84-g59dfed59dd2 [https://dl-cdn.alpinelinux.org/alpine/v3.22/community]
OK: 26332 distinct packages available
```

```
/ # apk add gdb
(1/14) Installing musl-dbg (1.2.5-r10)
(2/14) Installing libexpat (2.7.3-r0)
(3/14) Installing gmp (6.3.0-r3)
(4/14) Installing mpfr4 (4.2.1_p1-r0)
(5/14) Installing libffi (3.4.8-r0)
(6/14) Installing gdbm (1.24-r0)
(7/14) Installing mpdecimal (4.0.1-r0)
(8/14) Installing readline (8.2.13-r1)
(9/14) Installing sqlite-libs (3.49.2-r1)
(10/14) Installing python3 (3.12.12-r0)
(11/14) Installing python3-pycache-pyc0 (3.12.12-r0)
(12/14) Installing pyc (3.12.12-r0)
(13/14) Installing python3-pyc (3.12.12-r0)
(14/14) Installing gdb (15.2-r0)
Executing busybox-1.37.0-r18.trigger
OK: 68 MiB in 55 packages
```

Run GDB on core file

```
$ which clamscan
$ gdb -q /usr/bin/clamscan /dumps/core.clamscan.7
```

GDB confirms the core dump belongs to the `clamscan` process run with the specific arguments and that it terminated due to a **Segmentation fault (SIGSEGV)**.

```
/dumps # which clamscan
/usr/bin/clamscan
/dumps # gdb -q /usr/bin/clamscan /dumps/core.clamscan.7
Reading symbols from /usr/bin/clamscan...
[New LWP 7]
Core was generated by `clamscan --max-filesize=2048M --max-scansize=2048M --verbose /root/clam-cve.pdf'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  a_crash () at ./arch/x86_64/atomic_arch.h:108

warning: 108      ./arch/x86_64/atomic_arch.h: No such file or directory
(gdb)
```

Find the backtrace

```
(gdb) bt
```

```
(gdb) bt
#0  a_crash () at ./arch/x86_64/atomic_arch.h:108
#1  get_meta (p=p@entry=0xa005c064a005c064 <error: Cannot access memory at address 0xa005c064a005c064>) at src/malloc/mallocng/meta.h:131
#2  0x00007f5638d8eb9f in __libc_free (p=0xa005c064a005c064) at src/malloc/mallocng/free.c:105
#3  0x00007f563870c8ee in pdf_decodestream () from /usr/lib/libclamav.so.11
#4  0x00007f5638704f43 in pdf_extract_obj () from /usr/lib/libclamav.so.11
#5  0x00007f5638708f04 in cli_pdf () from /usr/lib/libclamav.so.11
#6  0x00007f563864dd4d in cli_scanpdf.constprop () from /usr/lib/libclamav.so.11
#7  0x00007f5638652e02 in cli_magic_scan () from /usr/lib/libclamav.so.11
#8  0x00007f56386561e6 in scan_common () from /usr/lib/libclamav.so.11
#9  0x00007f5638656cc6 in cl_scandesc_callback () from /usr/lib/libclamav.so.11
#10 0x000055b9dc1e2ae9 in scanfile ()
#11 0x000055b9dc1e5d8a in scanmanager ()
#12 0x000055b9dc1e1a6c in main ()
(gdb)
```

Here's a point-by-point summary of the backtrace:


- **Execution Flow:** The program started in `main` (Frame #12), proceeded through general scanning functions (`scanmanager`, `scanfile`, `scan_common`), detected a PDF (`cli_magic_scan`, `cli_scanpdf`), and entered the PDF parsing logic (`cli_pdf`, `pdf_extract_obj`).

- **The Critical Call:** `pdf_decodestream` (Frame #3), a function responsible for decoding PDF streams within `libclamav.so.11`, attempted to release memory by calling the C library's `__libc_free` function (Frame #2).
- **The Bad Pointer:** Crucially, `pdf_decodestream` passed an **invalid memory address** (`0xa005c064a005c064`) to `free`. This indicates a memory corruption issue occurred *before* or *within* `pdf_decodestream`.
- **The Crash Sequence:** The `free` function tried to process this bad pointer, leading it to call internal helper functions (`get_meta` at Frame #1). The attempt to access metadata using the invalid address ultimately caused the crash inside `a_crash` (Frame #0), resulting in the Segmentation Fault.
- **Importance of `pdf_decodestream`:** This function is **directly responsible for initiating the crash sequence** by calling `free` with the corrupted pointer. The root cause of the memory corruption likely lies within or before this function's execution during the PDF stream decoding process.

Disassemble pdf_decodestream

```
(gdb) set disassembly-flavor intel
(gdb) disas pdf_decodestream
```

```
0x00007f563870c8db <+3419>: mov     r15,QWORD PTR [rsp+0x10]
0x00007f563870c8e0 <+3424>: mov     QWORD PTR [rsp+0x28],r9
0x00007f563870c8e5 <+3429>: mov     rdi,QWORD PTR [r15+0x10]
0x00007f563870c8e9 <+3433>: call    0x7f56385fc7d8 <free@plt>
0x00007f563870c8ee <+3438>: mov     edx,DWORD PTR [r15+0x8]
0x00007f563870c8f2 <+3442>: mov     rsi,rbx
0x00007f563870c8f5 <+3445>: xor     eax,eax
0x00007f563870c8f7 <+3447>: lea     rdi,[rip+0x48570a]          # 0x7f5638b92008
0x00007f563870c8fe <+3454>: call    0x7f56385fa740 <cli_dbgmsg@plt>
```




Here's a breakdown of the assembly instructions:

- At `0x00007f563870c8e5`: The instruction reads an 8-byte value from memory at address `[r15+0x10]` and puts it into the `rdi` register. At this point the heap corruption has **already** occurred
- At `0x00007f563870c8e9`: The instruction calls the `free` function, using the value in `rdi` (the bad pointer) as the address to deallocate.

Find contents of pointer to free() call

```
(gdb) x/10wx $r15 + 0x10
```

```
(gdb) x/10wx $r15 + 0x10
0x7f56087d2190: 0xa005c064 0xa005c064 0xa005c064 0xa005c064
0x7f56087d21a0: 0xa005c064 0xa005c064 0xa005c064 0xa005c064
0x7f56087d21b0: 0xa005c064 0xa005c064
(gdb) █
```



This output directly shows that the 8 bytes read by `mov rdi, QWORD PTR [r15+0x10]` were `0xa005c064` followed by `0xa005c064`, forming the bad pointer `0xa005c064a005c064`. It confirms **heap corruption** occurred earlier, overwriting the expected valid pointer with this garbage pattern.

Locate source of heap corruption

In-Progress