APPLIED STEGANOGRAPHY

The Legal Disclaimer

Before we dive in, let's be clear: **Everything in this article is for educational purposes only**. These techniques should only be used in authorized penetration tests, CTF competitions, or lab environments where you have explicit permission.

Using these methods against systems you don't own or don't have authorization to test is illegal.

Picture this: You've crafted the perfect payload. It's a beautiful reverse shell, ready to give you access to your target system. But there's one problem every antivirus on the planet recognizes it instantly. You have to find a way to evade from these detection systems in order to create something that really works.

What if I told you that with a few simple tricks, you could get that same payload past almost every single one of those engines? Not by exploiting some zero-day vulnerability or writing complex polymorphic code, but by simply hiding it inside a cat picture?

In this article, I'm going to walk you through three practical techniques I've tested for hiding payloads from antivirus detection.

Why Should You Care?

If you're a pentester, red teamer, or just someone learning offensive security, you know the frustration: You've compromised a system, but your C2 agent gets deleted within seconds. Or your phishing payload gets caught before it even executes. Or your custom tool that took hours to write gets flagged instantly.

Steganography isn't about making malware "undetectable forever which is not realistic. But it's about buying yourself time, bypassing initial defenses, and understanding how detection works so you can work around it.

And if you're on the blue team? Understanding these techniques is crucial. You can't defend against what you don't understand.

All tests were conducted in a controlled lab environment:

- **Attacker Machine:** Kali Linux (latest)
- **Target Machine:** Windows 10 / Ubuntu 20.04
- **Detection Platform:** VirusTotal – Metadefender

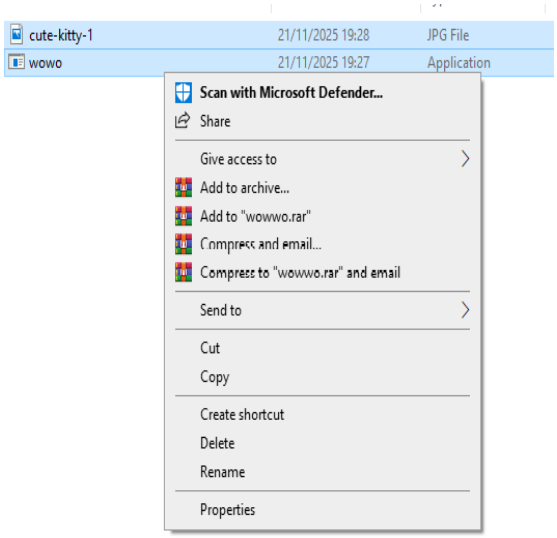
Lets start with windows payload.

1) Create our payload with msfvenom

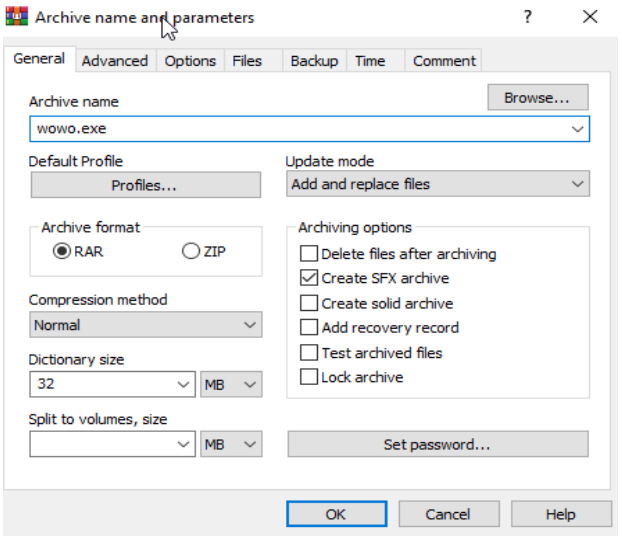
2) Uploading the plain payload to Metadefender for detection analysis. This will show us how many antivirus engines recognize our unobfuscated payload.

3) Now lets obsfucate it in our windows system with some basic operations.

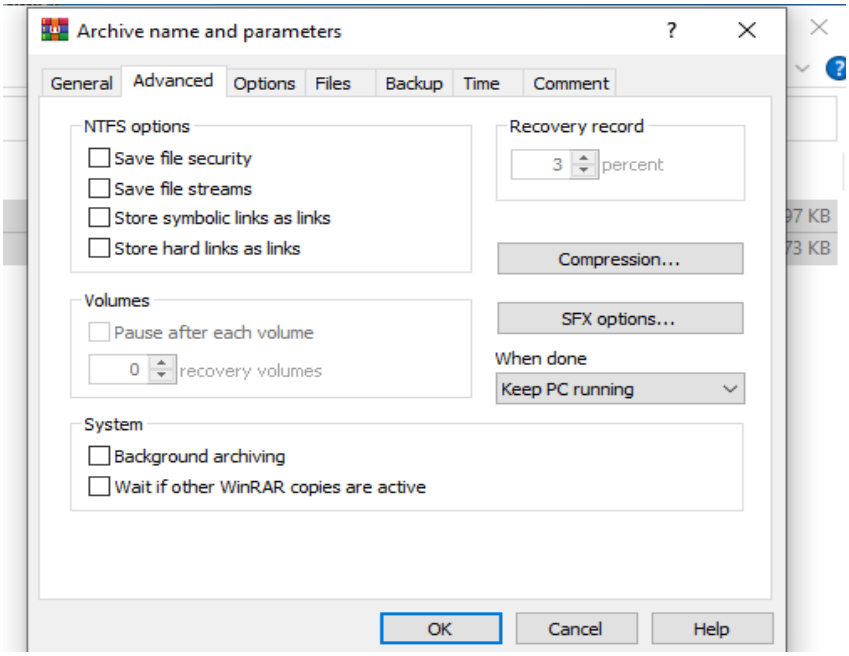
1- Starting the SFX archive creation process. We select both our payload executable and a legitimate image file, then access WinRAR's archive options.



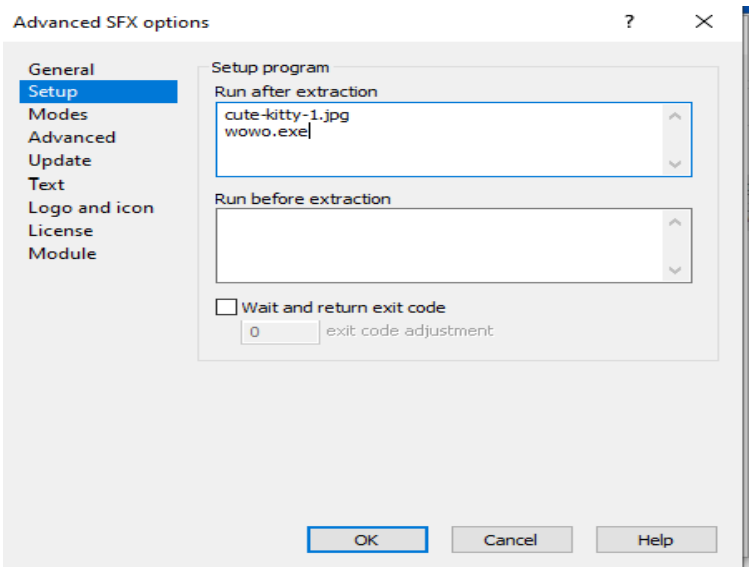
2-Enabling the 'Create SFX archive' option. Self-extracting archives allow automatic execution upon opening, which we'll use to deliver our payload.



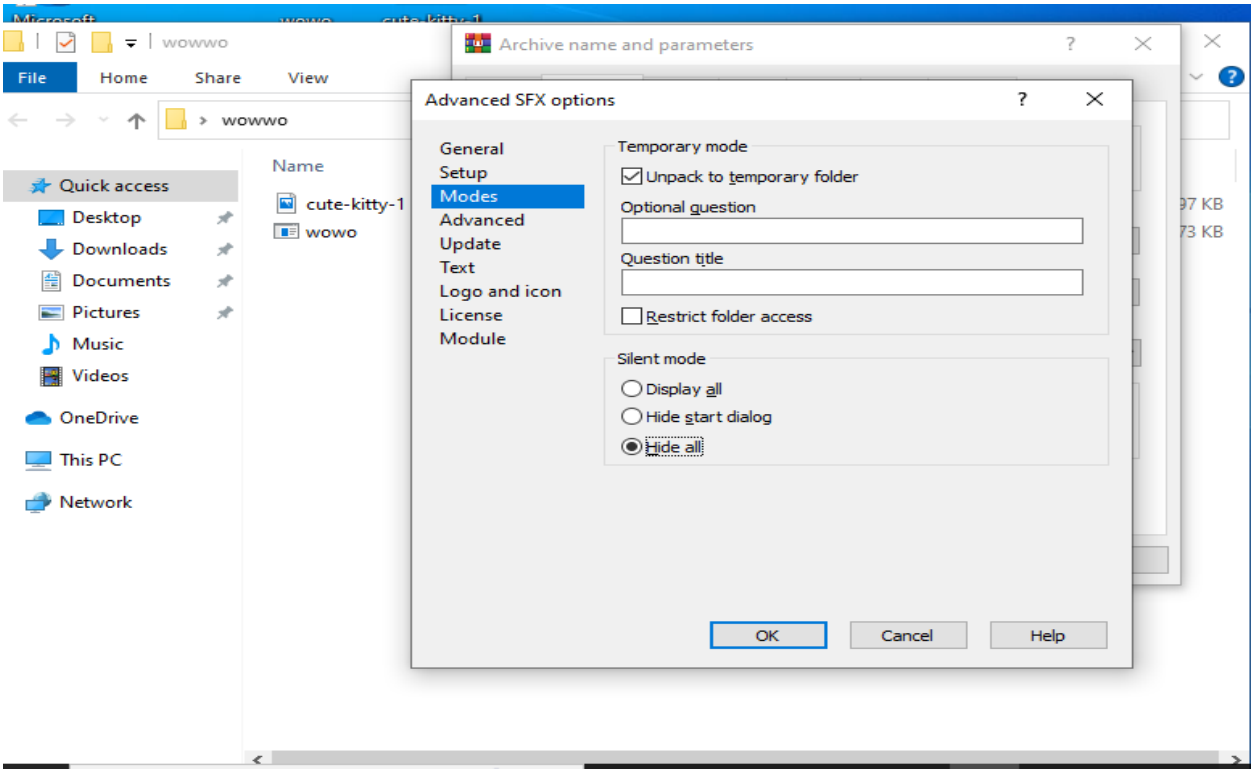
3- Accessing the SFX options from the Advanced tab. Here we configure the self-extracting behavior and stealth settings.



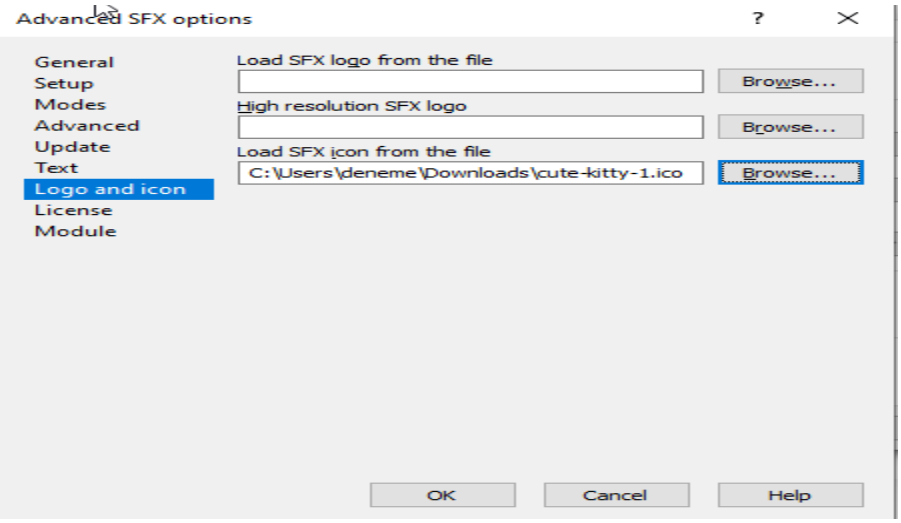
4- Configuring the execution order in the Setup tab. We set the image to open first, followed by the payload. This maintains the illusion of a harmless image file.



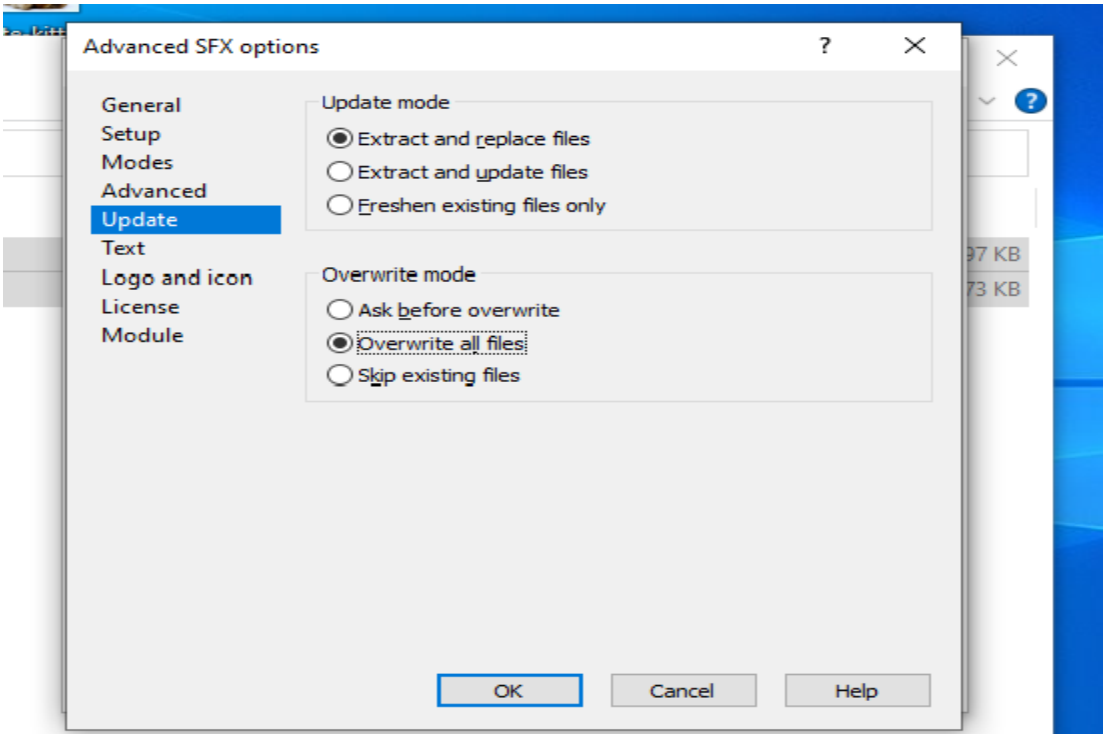
5- Setting the Modes tab to ‘Unpack to temporary folder’ and ‘Hide all’. These settings ensure the extraction process is completely invisible for the victim.



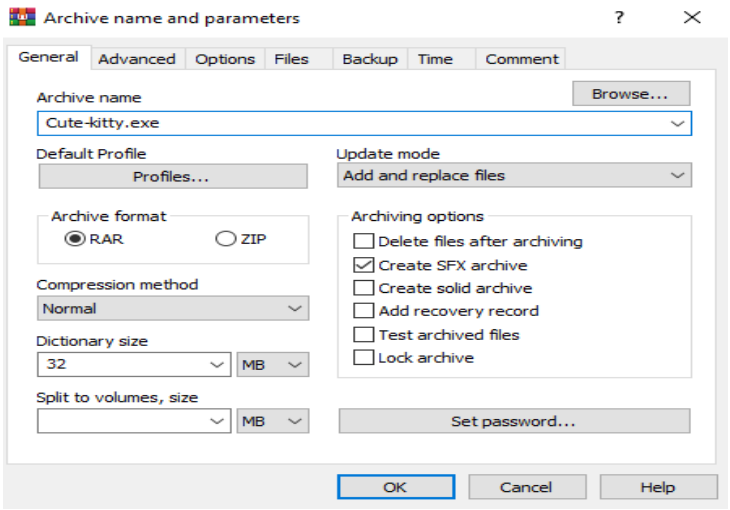
6- Select a custom icon that resembles an image file. This visual disguise is important.



7-Setting the Update tab to 'Overwrite all files'mode. This prevents any dialog boxes that might alert during the execution.

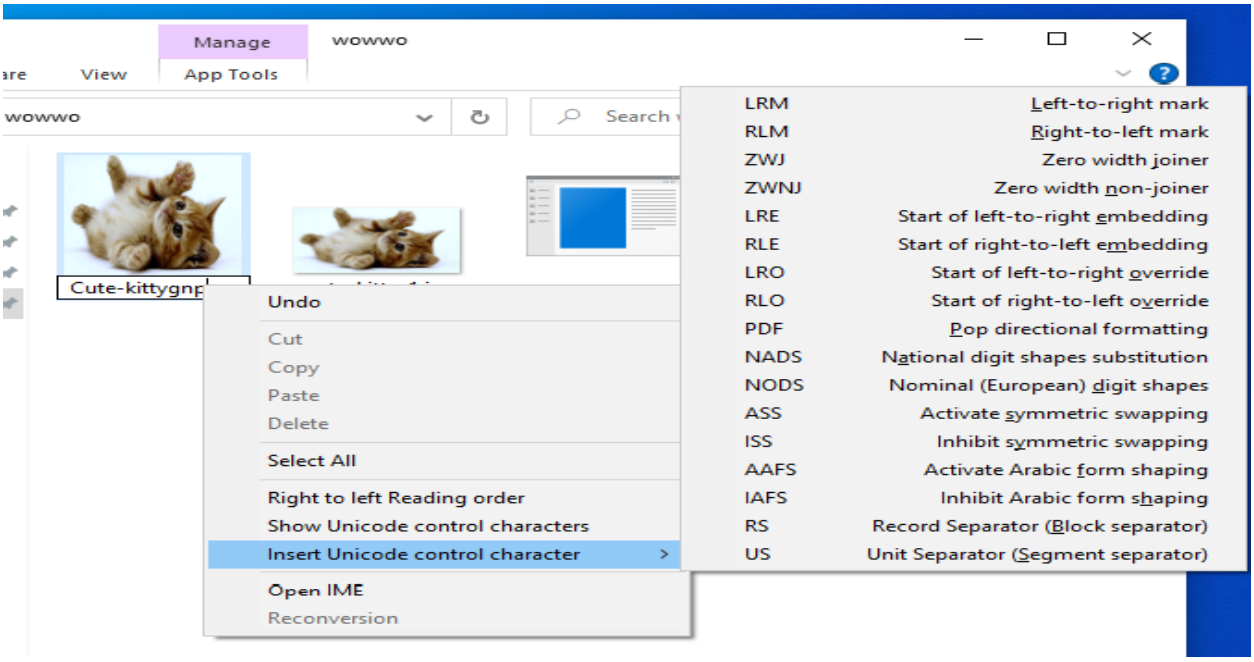


8- Naming the archive file. At this stage, it still has the .exe extension, which we'll disguise using the Unicode RLO technique in the next steps.

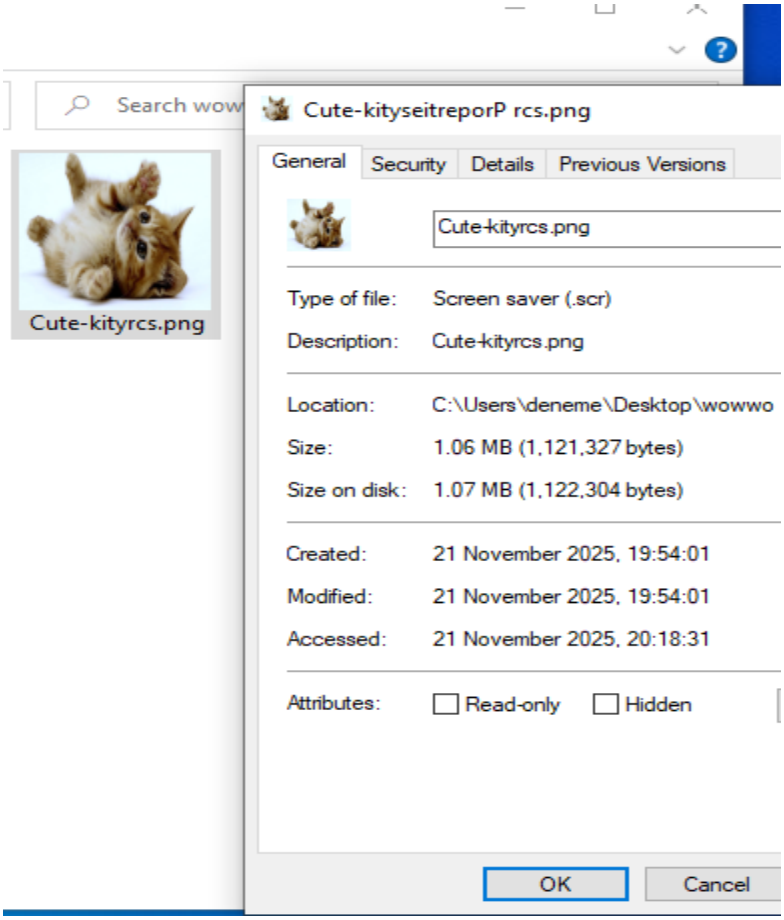


9- The archive has been created successfully. The file appears with our chosen icon

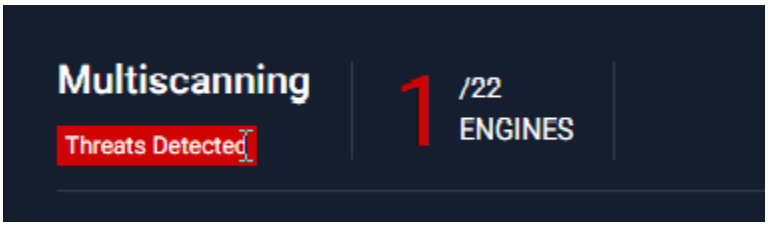
10- Applying the Unicode Right-to-Left Override (RLO) trick. We add the reverse of our desire extension, then insert the RLO character to make '.scr' appear as '.png'.



12- After applying the RLO, the file displays with '.png' extension but Windows recognizes it as a Screen Saver (.scr) file.



4) And the final detection result shows a dramatic reduction in detection rate compared to our plain payload.



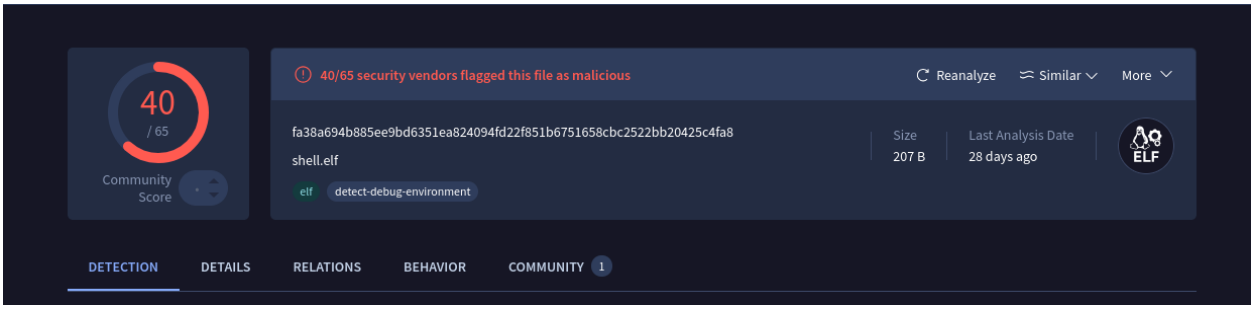
Note: While Unicode RLO-based extension spoofing is still technically effective, modern email gateways and EDR solutions often flag such files as high-risk, making this technique most viable only in low-maturity or legacy environments.

We've successfully demonstrated how Windows payloads can evade detection using SFX archives and extension spoofing. Now let's shift our focus to Linux payloads, where we'll explore

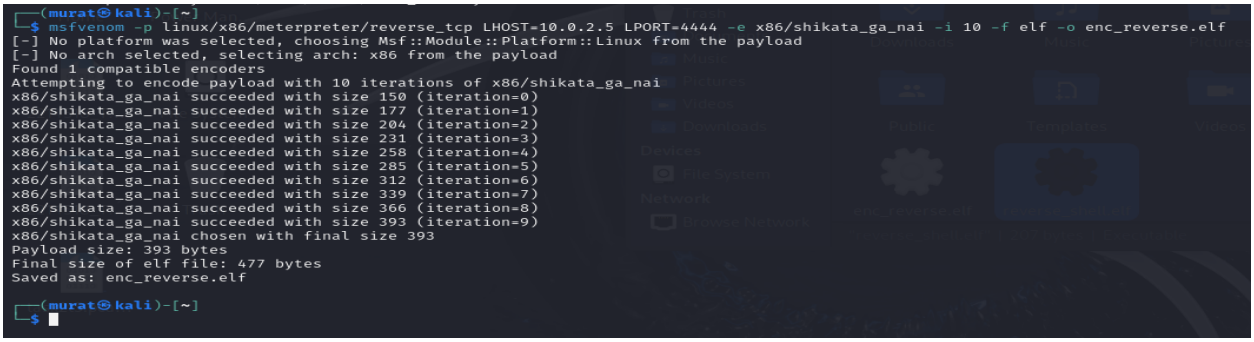
two different approaches. First, we'll use Metasploit's built-in polymorphic encoder to scramble our payload's signature. Then, we'll take things further with true steganography ‘literally hiding our executable inside an image file’.

1) Lets create our Linux payload with msfvenom

2)) Uploading the plain Linux payload to VirusTotal. The baseline detection shows how many antivirus engines recognize the unencoded Meterpreter payload. Detection rate: 40/65

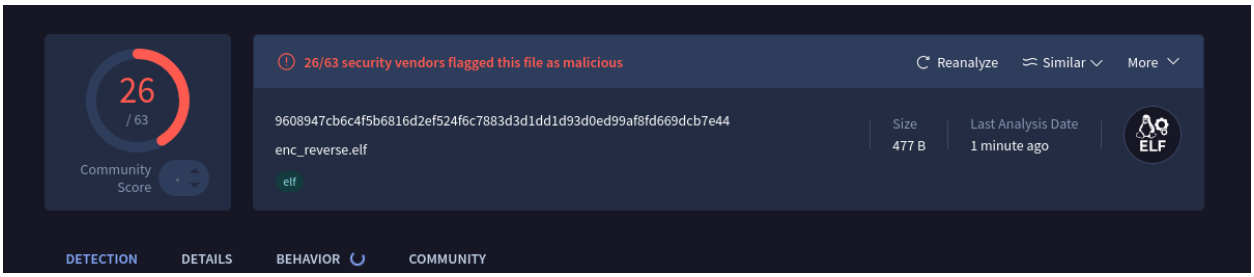


3) Applying Shikata Ga Nai encoder with 10 iterations. To create a polymorphic payload that generates a unique signature with each encoding

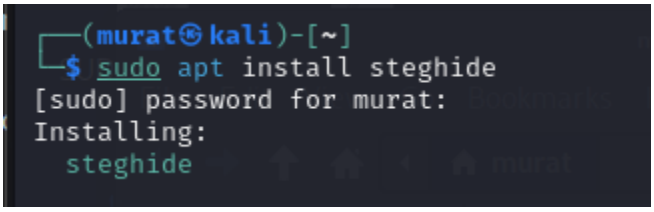


4) Detection results after Shikata Ga Nai encoding. The polymorphic encoder reduced detection from 40/65 to 26/63 engines. While effective, since Shikata Ga Nai is a very old method most engines still detect the encoded payload.

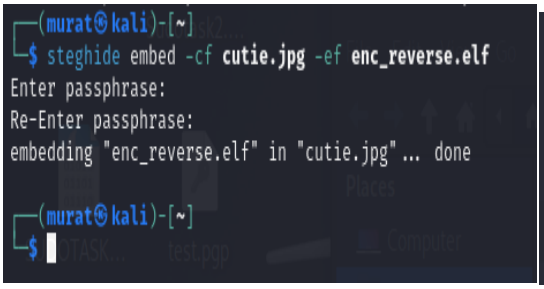
Note: Today, Shikata Ga Nai should be considered a baseline obfuscation technique, not a standalone evasion method.

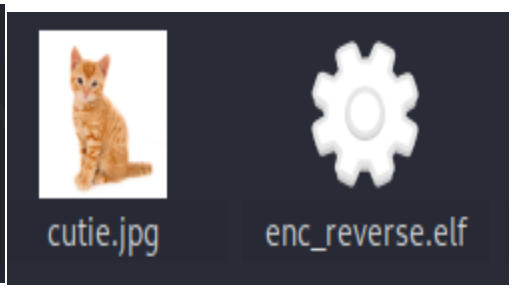


5) Installing Steghide on Kali Linux. This steganography tool will allow us to embed our payload inside an image file using LSB (Least Significant Bit) substitution.



6) Embedding the payload into a JPEG image using Steghide. We specify the cover file (innocent image), the file to embed (our payload), and set a password for extraction. The process completes successfully.



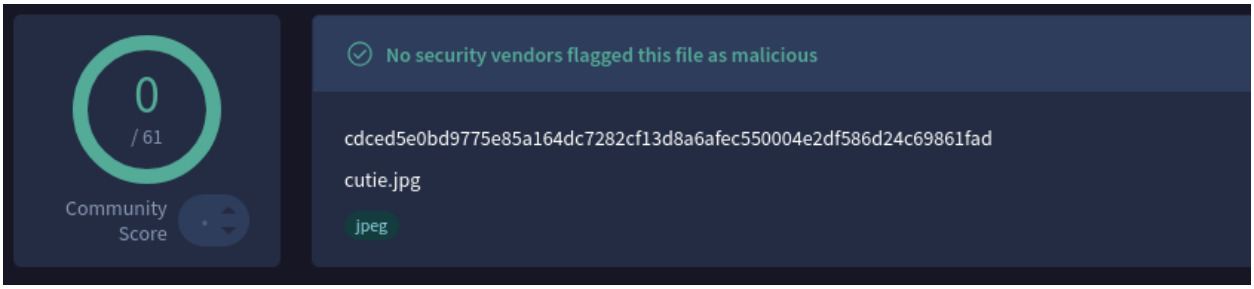


7) Visual comparison of original vs stego image. Both images appear identical to the human eye. Interestingly, the stego image is actually smaller due to JPEG re-compression during embedding—making detection even harder.

8) Final detection results: Uploading the stego image to VirusTotal reveals 0/60 detections.

This indicates successful evasion of static, signature-based detection mechanisms, as antivirus engines classify the file as a legitimate JPEG image and do not inspect the embedded payload.

It is important to note that this does not imply evasion from behavioral analysis, sandbox execution, or EDR solutions, which could still detect malicious activity once the payload is extracted and executed.



Conclusion

What We've Learned

We explored three practical steganography and obfuscation techniques:

Windows - SFX Archive Method:

- Combines payload with legitimate files in self-extracting archives
- Uses Unicode RLO to disguise file extensions
- Achieves evasion against static, signature-based detection mechanisms
- **Linux - Shikata Ga Nai Encoder:**
- Polymorphic encoding creates unique payload signatures

- Multiple iterations increase obfuscation effectiveness
- Reduces but doesn't eliminate detection
- Good first layer before additional hiding techniques

Linux - Steghide Image Steganography:

- Embeds payloads inside image files using LSB substitution
- Achieves evasion against static, signature-based detection mechanisms
- Requires manual extraction and execution
- Ideal for covert data transfer and payload delivery

Key Takeaways

1. **Different techniques serve different purposes** - SFX for execution convenience, encoding for signature evasion, steganography for transfer stealth.
2. **Detection evasion ≠ execution evasion** - These techniques bypass file scanning, but behavioral analysis during execution is another challenge.
3. **Layering increases effectiveness** - Combining techniques (encode → hide → deliver) provides defense in depth against detection.
4. **Trade-offs exist** - Steganography offers the best evasion but requires extraction steps. SFX is immediately executable but less stealthy.

Defense Perspective

Understanding offensive techniques improves defensive capabilities:

- **File type validation** - Verify files by content, not extension
- **Behavioral monitoring** - Watch what files **do**, not just what they **are**
- **Steganalysis tools** - Detect hidden data in images
- **User awareness** - Train users to question unexpected files
- **Sandbox analysis** - Test suspicious files in isolated environments

The security community benefits when both offensive and defensive knowledge advance together.

Remember: this knowledge is for **authorized security testing and education only**. Using these methods against systems without permission is illegal and unethical.

Stay curious. Stay ethical