

# Mã độc

## Chương 7. Các kỹ thuật chống phân tích

# Mục tiêu

- Giới thiệu, phân tích một số kỹ thuật chống phân tích thường gặp trong mã độc

# Tài liệu tham khảo

**[1] Michael Sikorski, Andrew Honig, 2012, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press, (ISBN: 978-1593272906).**

**[2] Sam Bowne, Slides for a college course at City College San Francisco,  
[https://samsclass.info/126/126\\_S17.shtml](https://samsclass.info/126/126_S17.shtml)**

# Nội dung

- 1. Nén**
- 2. Mã hóa**
- 3. Làm rối mã**
- 4. Một số cơ chế khác**

# Nội dung

**1. Nén**

**2. Mã hóa**

**3. Làm rối mã**

**4. Một số cơ chế khác**

# Nén

- ❑ **Packer (nén) là một kiểu chương trình nén hoặc che giấu file thực thi (executable file).**
- ❑ **Giảm kích thước của file, làm cho việc tải file nhanh hơn.**

# Nén

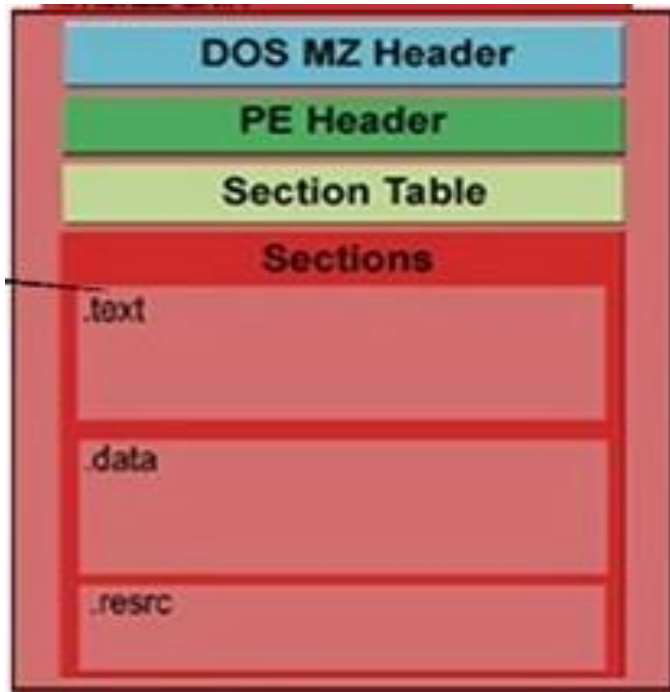
- ❑ Packer nén, mã hóa, lưu hoặc dấu code gốc của chương trình, tự động bổ sung một hoặc nhiều section, sau đó sẽ thêm đoạn code Unpacking Stub và chuyển hướng Entry Point (EP) tới vùng code này.
- ❑ Một file không đóng gói (Nonpacked) sẽ được tải bởi OS.

# Nén

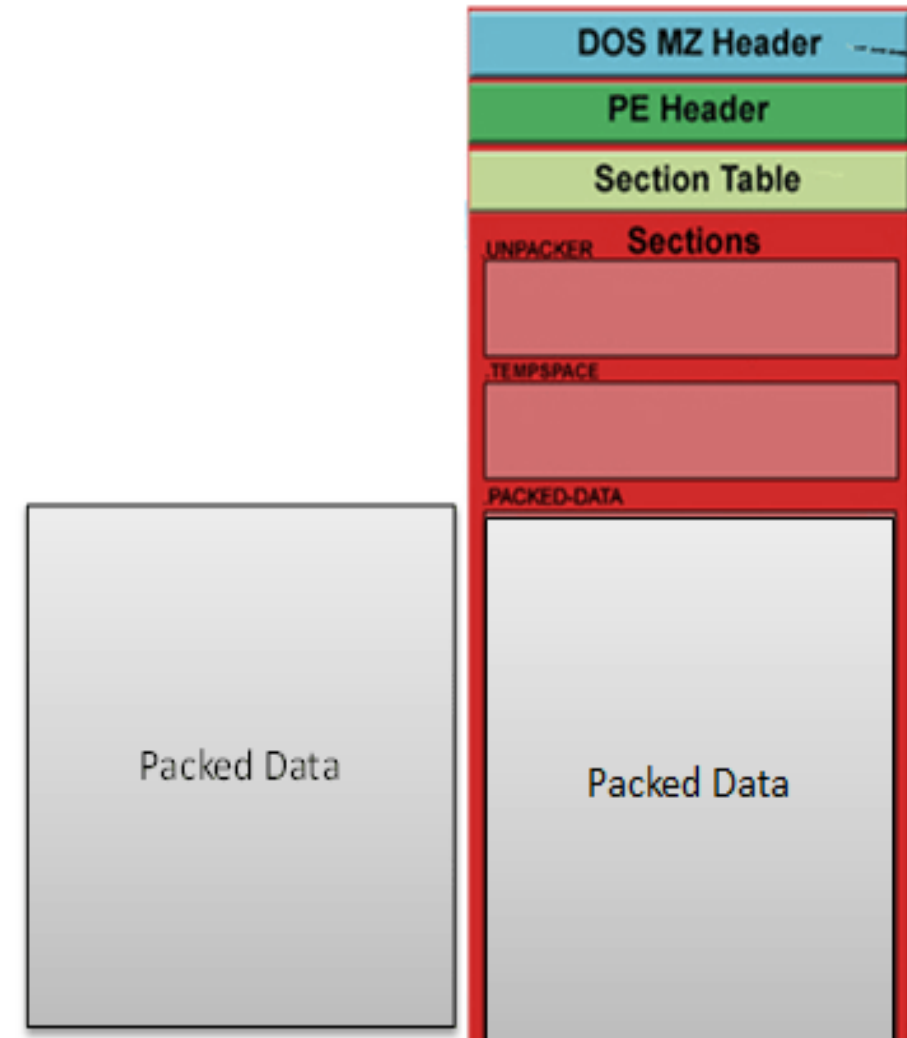
- ❑ File đã đóng gói: Unpacking Stub sẽ được tải bởi OS, sau đó chương trình gốc sẽ tải Unpacking Stub.
- ❑ Code EP của file thực thi sẽ trở tới Unpacking Stub thay vì trở vào mã gốc.
- ❑ Trong môi trường DOS, chương trình sẽ kiểm tra DOS header và nếu hợp lệ sẽ thực thi DOS Stub. Bình thường file chạy trên Windows thì 2 trường này có thể bỏ qua.



# Nguyên lí hoạt động của Packer

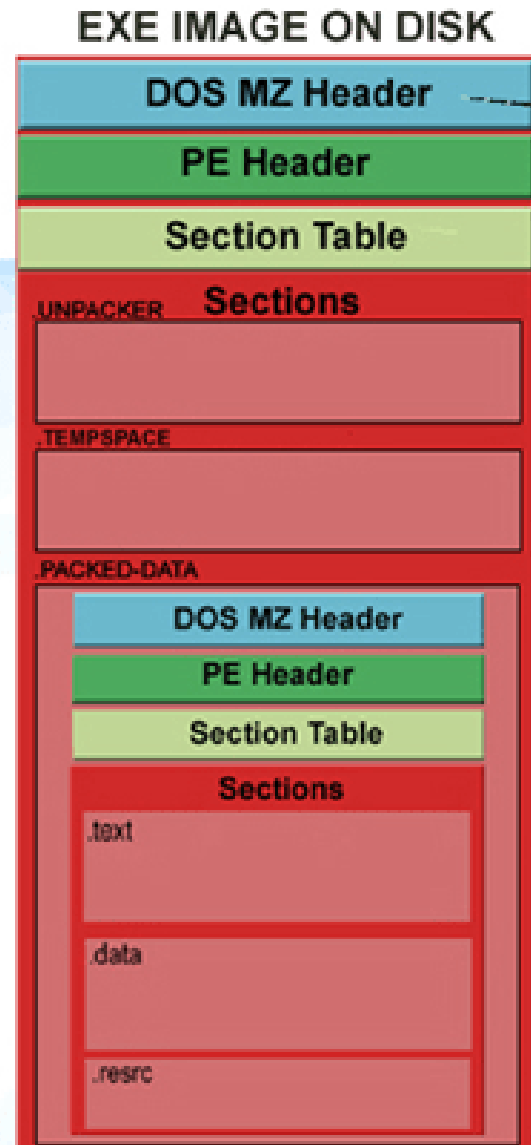


File ban đầu

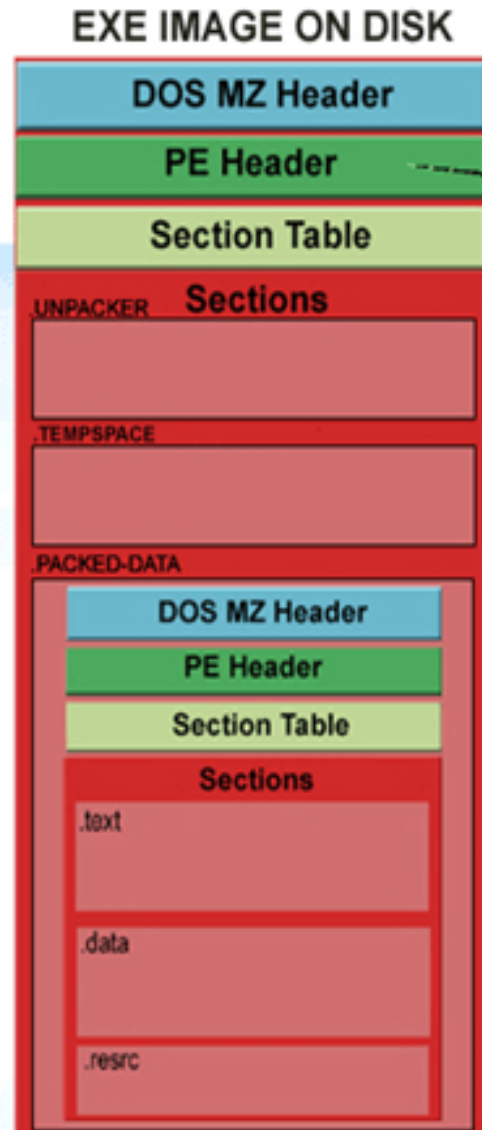


Thêm vào một hay nhiều section  
do packer tự tạo ra

# Thực thi tệp đã nén

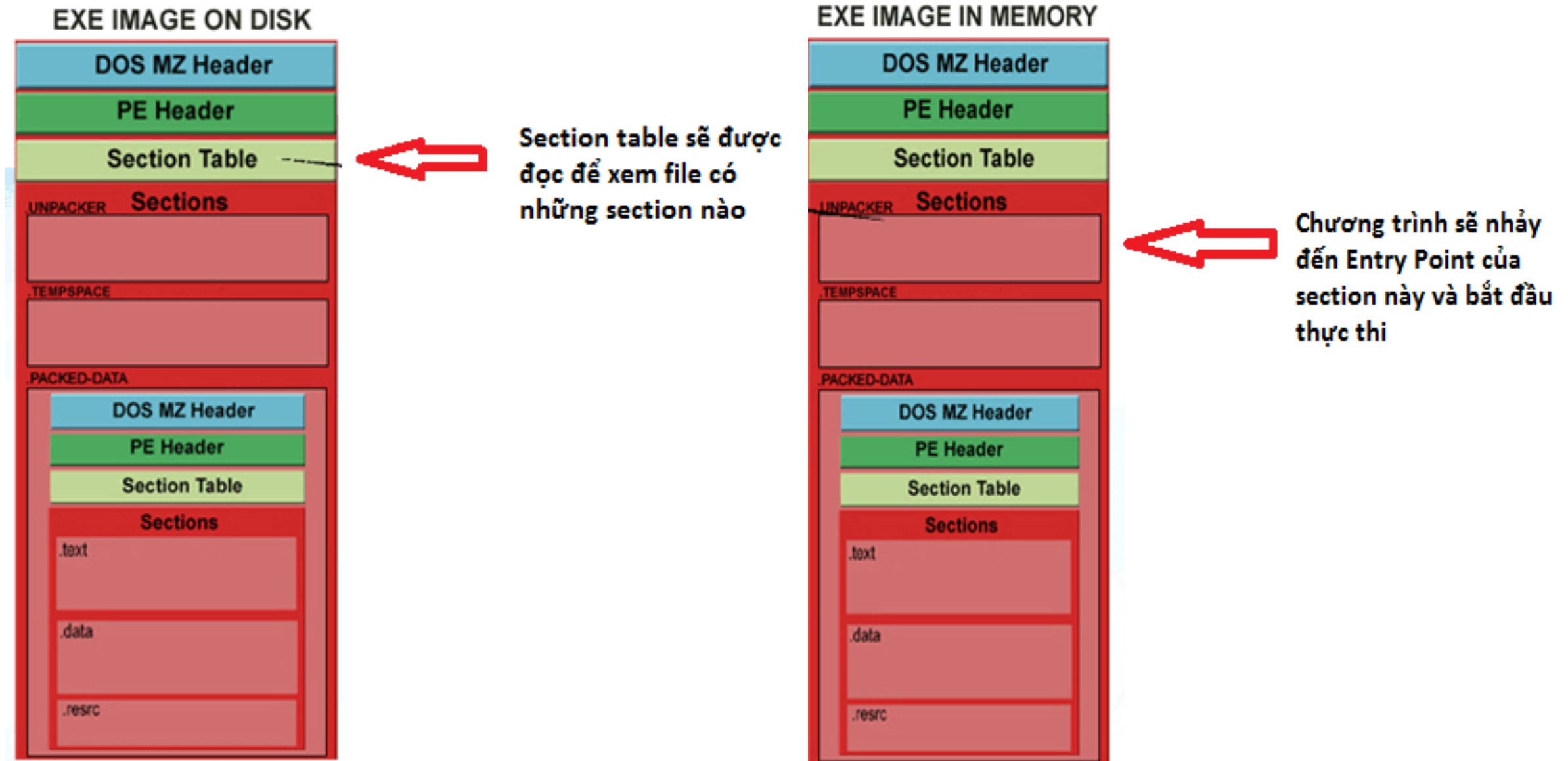


Khi không chạy  
trong môi trường  
DOS thì phần này sẽ  
bị bỏ qua



Tiếp theo chương trình sẽ  
thực thi PE header

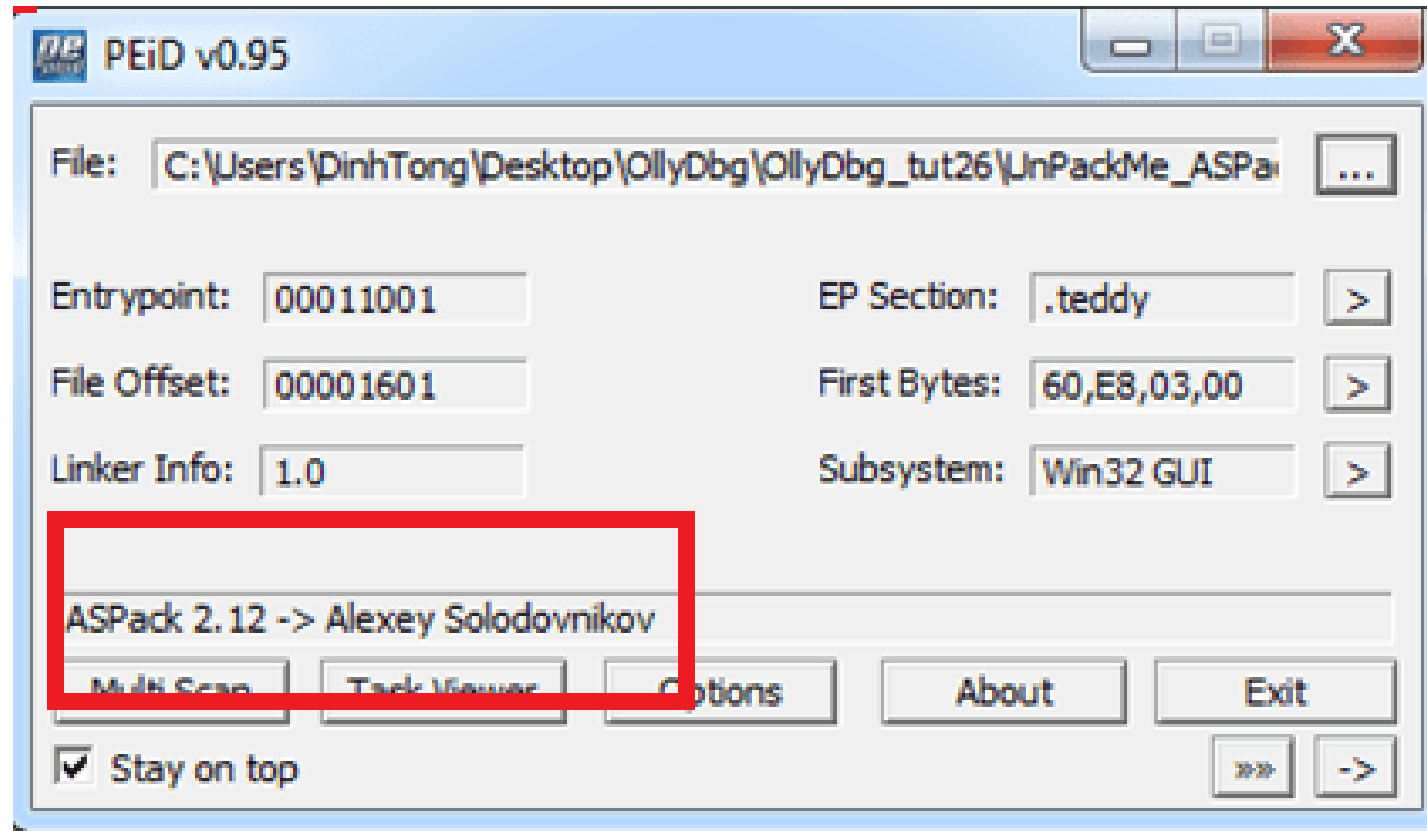
# Thực thi tệp đã nén



# Thực thi tệp đã nén

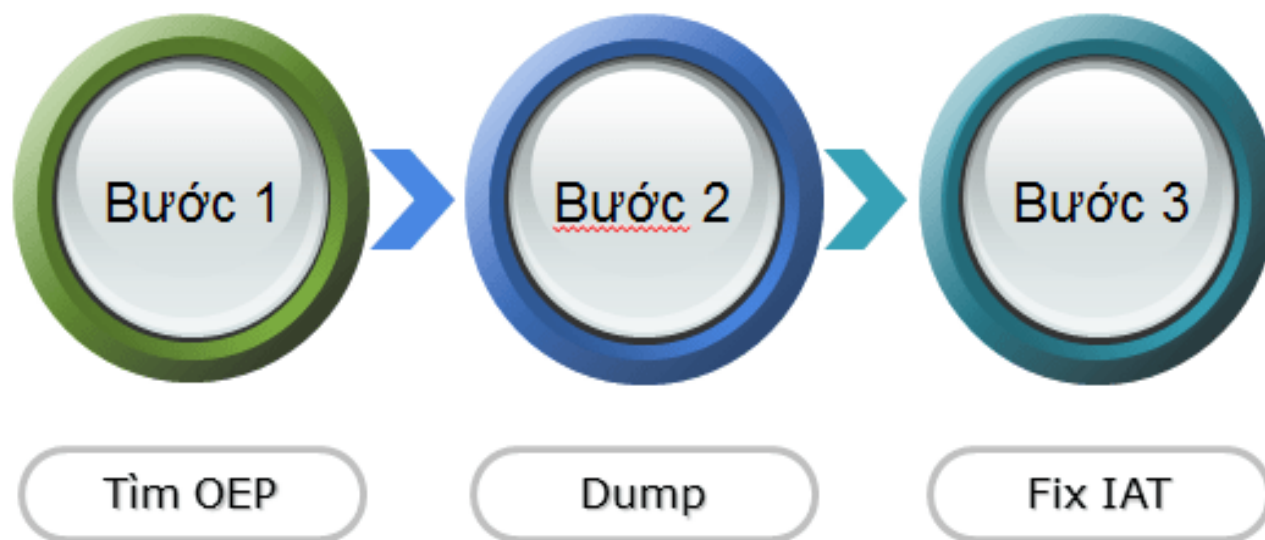
- ❑ Nhảy đến Entry Point của section UNPACKER, lưu lại toàn bộ giá trị của các thanh ghi bằng lệnh PUSHAD.
- ❑ Tính toán lại Import Table.
- ❑ Khôi phục lại giá trị các thanh ghi đã được lưu trong Stack bằng lệnh POPAD.
- ❑ Nhảy đến Origin EntryPoint và thực thi như file lúc chưa bị đóng gói.

# Nhận biết một tệp tin bị nén



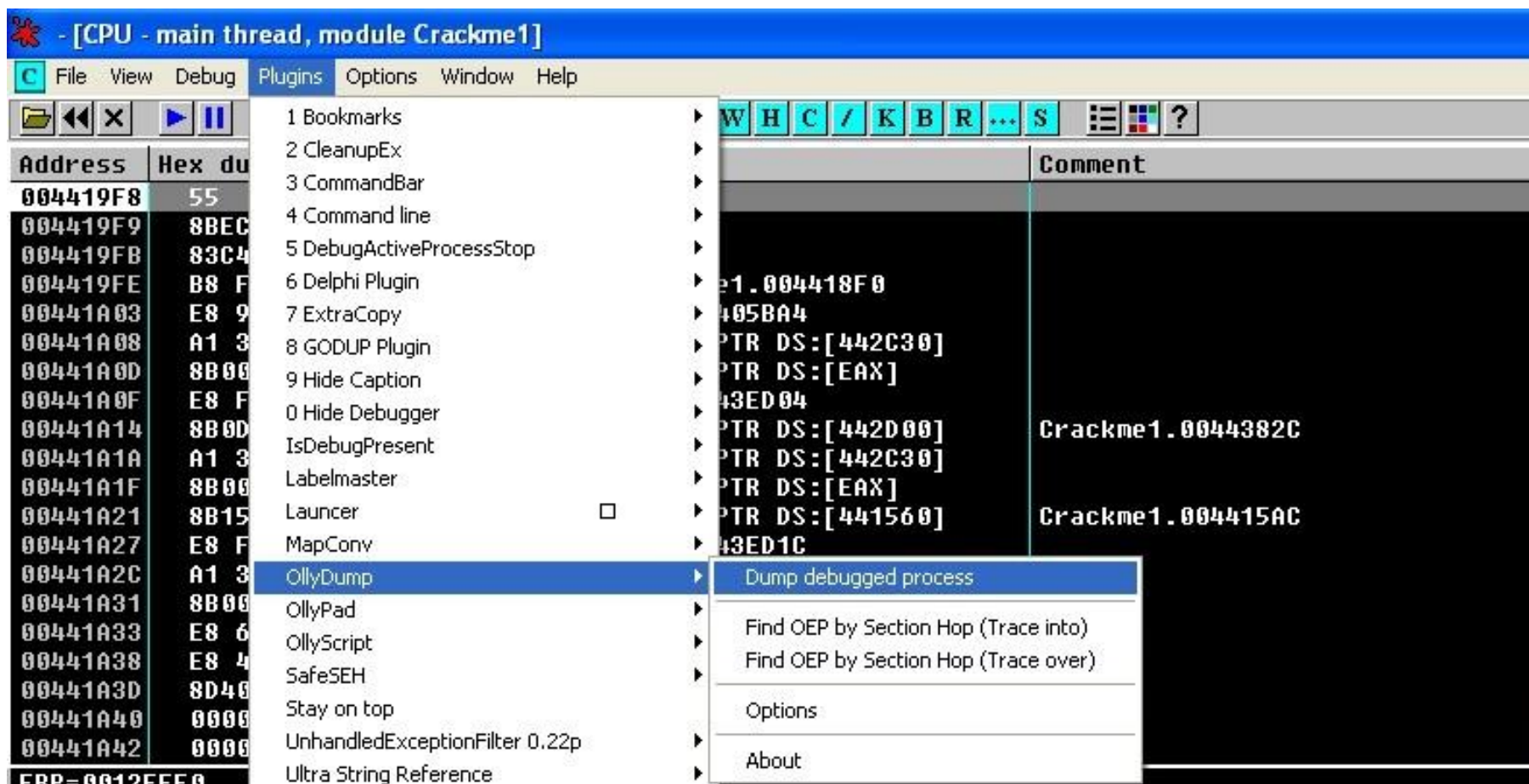
# Giải nén

## ❑ Các bước cơ bản để thực hiện giải nén



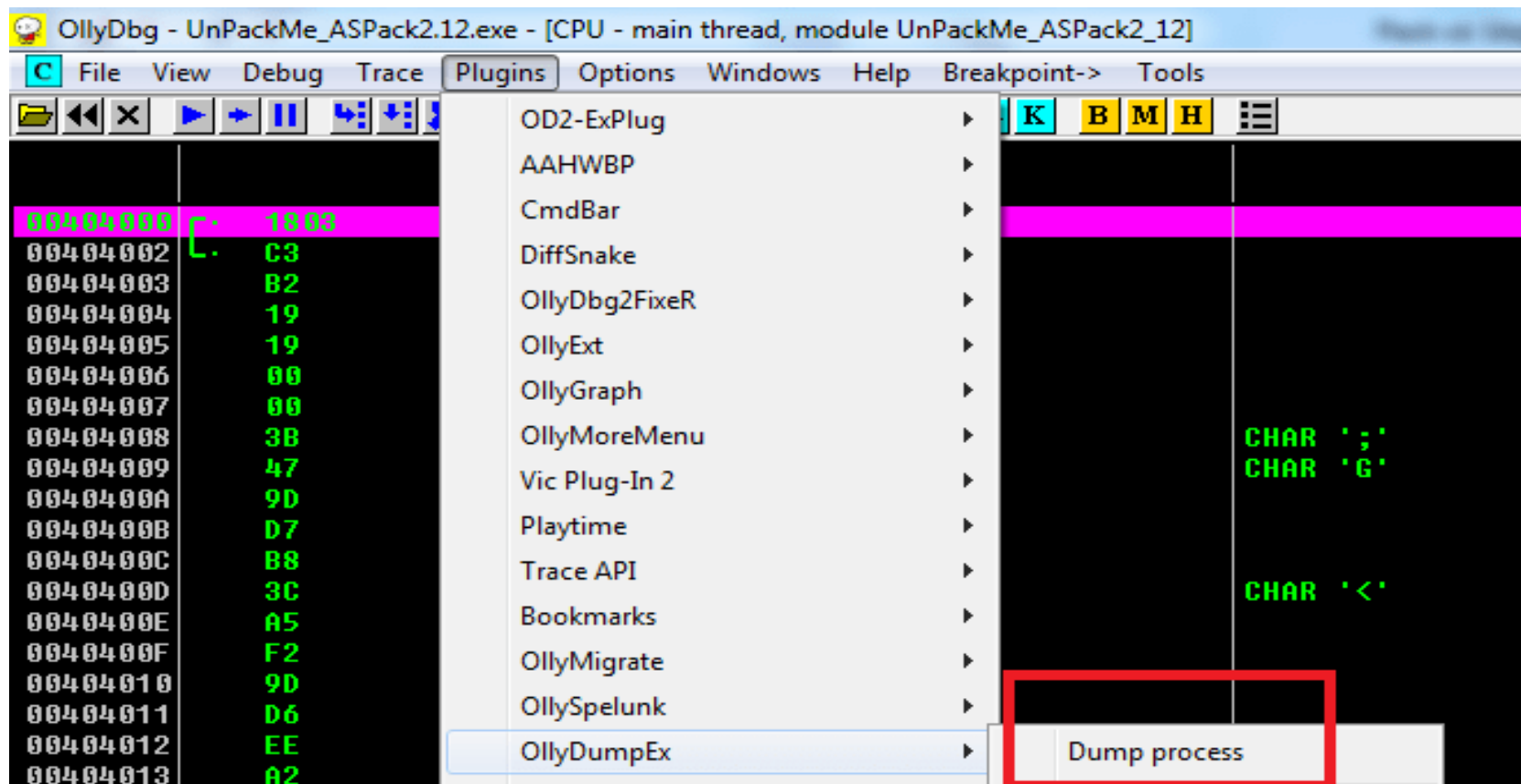
# Tìm OEP

❑ Original entry point là nơi mà chương trình gốc thực sự bắt đầu thực thi.



# Dump file

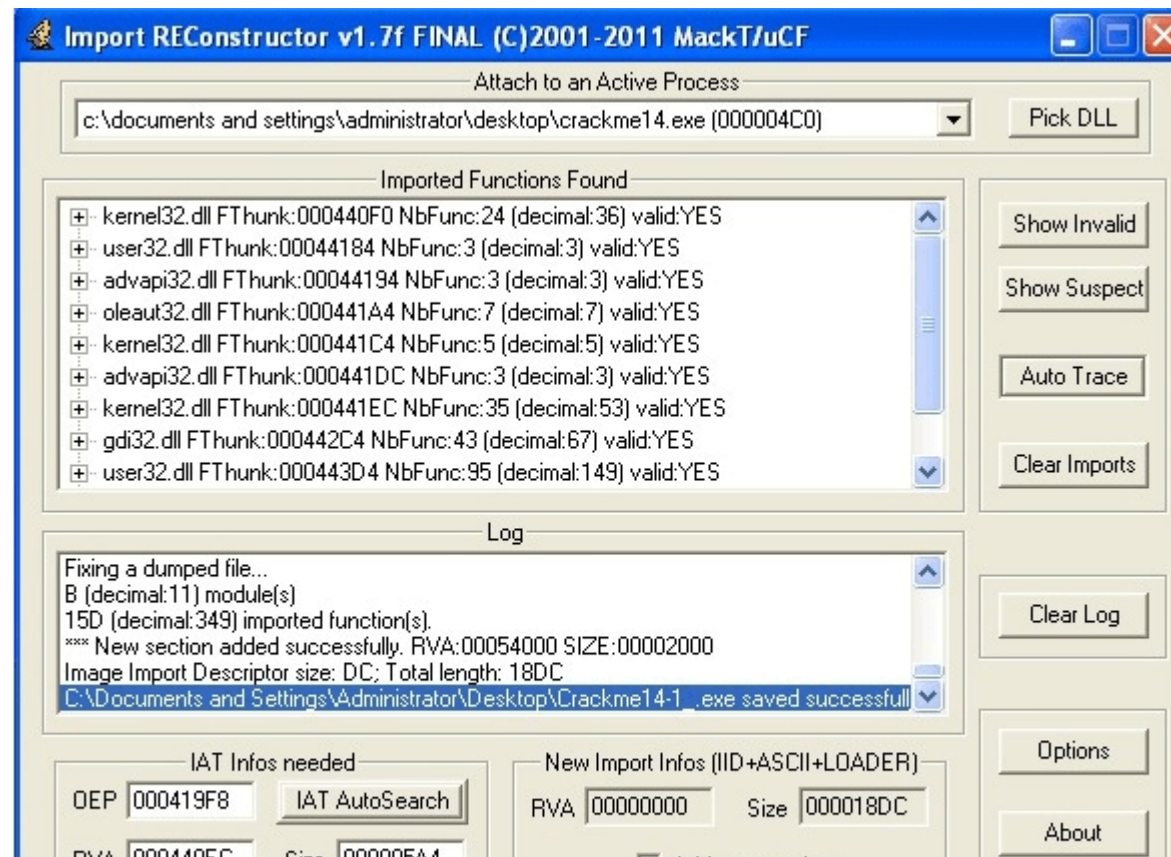
❑ Sau khi nhảy đến OEP ta sẽ tiến hành dump file. Mục đích của việc này là fix lại các section và import table như file ban đầu trước khi bị đóng gói.





# Fix IAT

❑ Kiểm tra file xem còn các cơ chế Anti hoặc ngăn chặn việc thực thi hay không rồi tiến hành chỉnh sửa. Đảm bảo file sau unpack thực thi bình thường



# Nội dung

**1. Nén**

**2. Mã hóa**

**3. Làm rối mã**

**4. Một số cơ chế khác**

# Mã hóa

**Lý do mã độc sử dụng các biện pháp mã hóa:**

- ☐ Che giấu thông tin cấu hình, chẳng hạn như C&C Domain.
- ☐ Lưu thông tin vào tệp tin trước khi đánh cắp
- ☐ Che giấu mã độc bên trong một công cụ hợp pháp
- ☐ Ẩn các chuỗi bị nghi ngờ

# Mã hóa

- ☐ Thuật toán mã hóa đơn giản.
- ☐ Thuật toán mã hóa mạnh
- ☐ Thuật toán mã hóa tùy chỉnh

# Mã hóa

- ☐ Thuật toán mã hóa đơn giản
- ☐ Thuật toán mã hóa mạnh
- ☐ Thuật toán mã hóa tùy chỉnh

# Thuật toán mã hóa đơn giản

- ❑ Kích thước nhỏ, phù hợp với các môi trường bị ràng buộc như khai thác bằng shellcode
- ❑ Ít ảnh hưởng đến hiệu năng
- ❑ Gây khó khăn trong việc phát hiện, tuy nhiên không thể qua mặt được các nhà phân tích có kỹ năng tốt

# Mã Caesar

❑ Dịch chuyển từng chữ cái về trước 3 vị trí trong bảng chữ cái alphabet

ABCDEFGHIJKLMNOPQRSTUVWXYZ  
DEFGHIJKLMNOPQRSTUVWXYZABC

❑ Ví dụ

ATTACK AT NOON  
DWWDFN DW QRRQ

# XOR


- ❑ Sử dụng một khóa để mã hóa
- ❑ Sử dụng một bit của dữ liệu và một bit của khóa trong một thời điểm
- ❑ Ví dụ: Encode Encode HI với khóa 0x3C  
HI = 0x48 0x49 (ASCII encoding)

Data:	0100 1000 0100 1001
Key:	0011 1100 0011 1100
Kết quả:	0111 0100 0111 0101

0	xor	0	=	0
0	xor	1	=	1
1	xor	0	=	1
1	xor	1	=	0



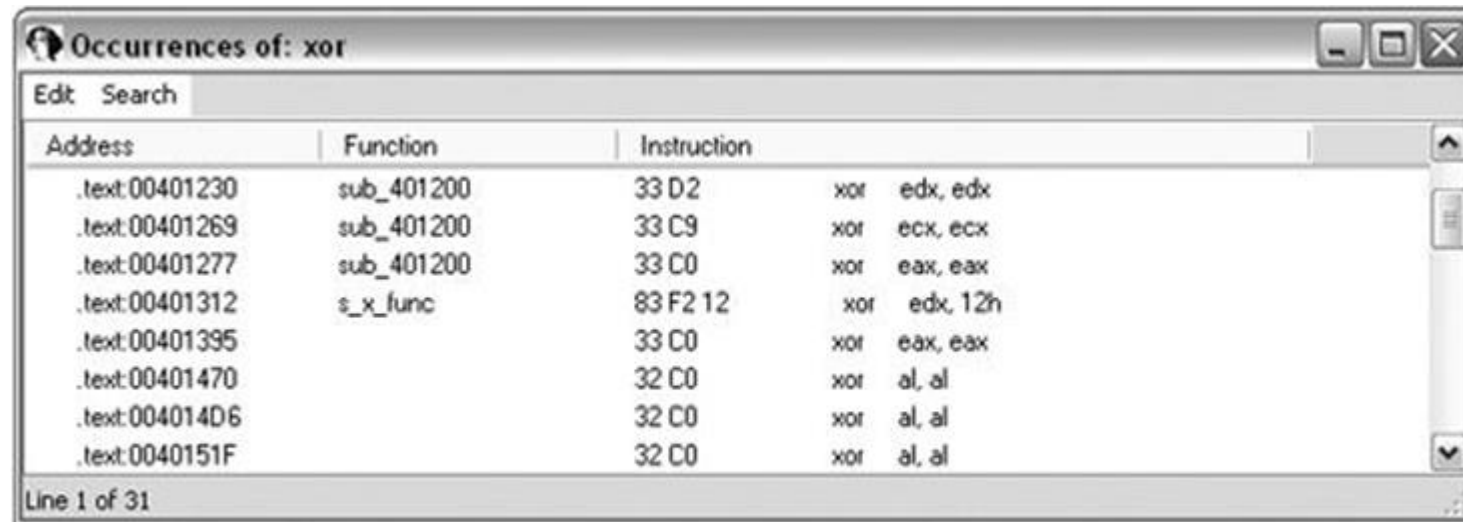
# XOR

A	T	T	A	C	K		A	T		N	O	O	N
0x41	0x54	0x54	0x41	0x43	0x4B	0x20	0x41	0x54	0x20	0x4E	0x4F	0x4F	0x4E
													
}	h	h	}	DEL	W	FS	}	H	FS	r	s	s	r
0x7d	0x68	0x68	0x7d	0x7F	0x77	0x1C	0x7d	0x68	0x1C	0x72	0x71	0x71	0x72

*The string ATTACK AT NOON encoded with an XOR of 0x3C  
(original string at the top; encoded strings at the bottom)*

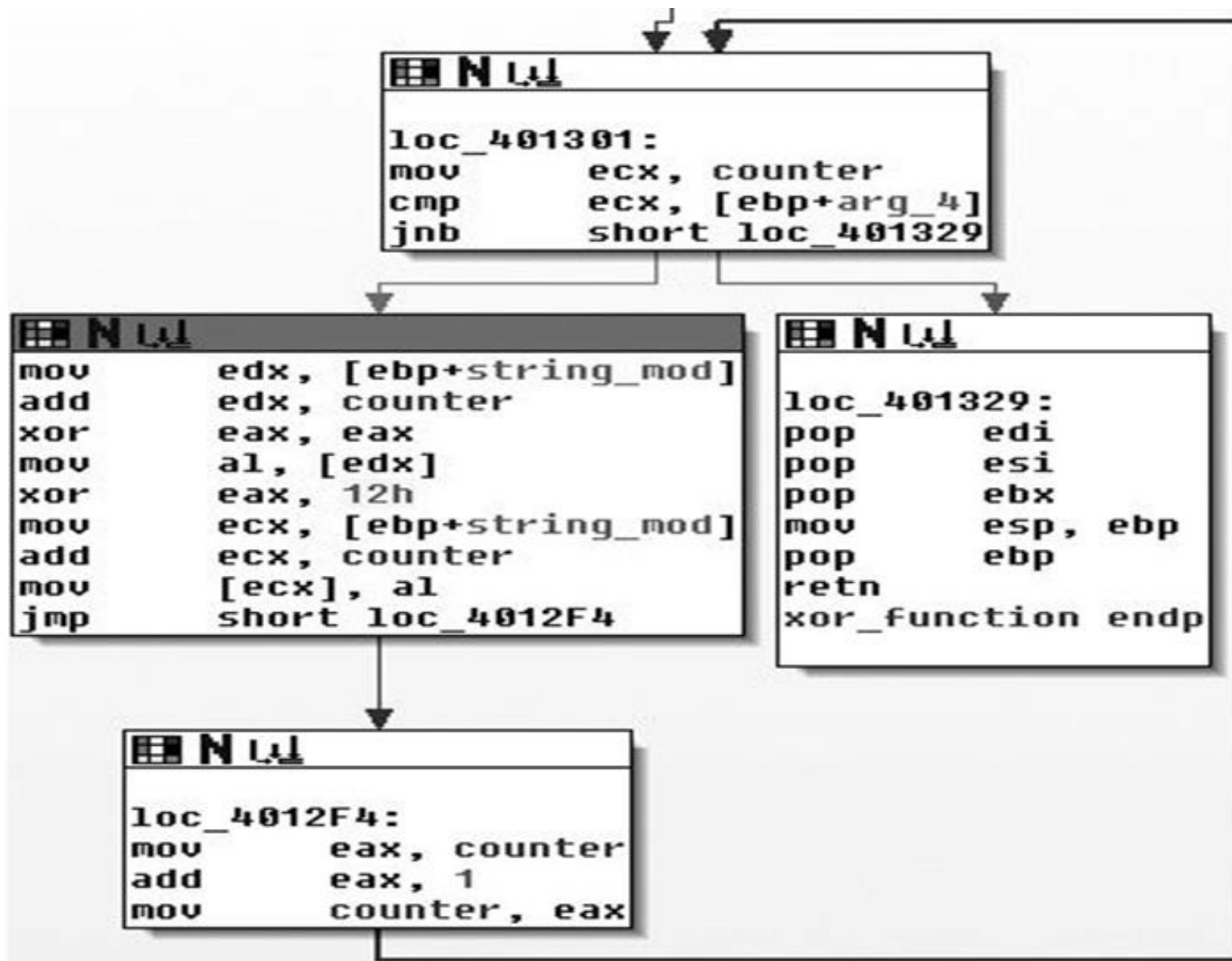
# Xác định vòng XOR trong IDA Pro

- ❑ Các vòng lặp với lệnh XOR bên trong
- ❑ Bắt đầu với “IDA View” (xem code)
- ❑ Click Search, Text
- ❑ Nhập xor và tìm tất cả các lần xuất hiện



*Searching for XOR in IDA Pro*

# Xác định vòng XOR trong IDA Pro



# Base64

- ❑ Chuyển đổi 6 bits thành một ký tự trong bảng chữ cái 64 ký tự
- ❑ Sử dụng các khối 3 byte (24 bits)
- ❑ Chia thành 4 trường 6-bits
- ❑ Chuyển từng trường thành base64

# Base64

*Part of raw email message showing Base64 encoding*

```
Content-Type: multipart/alternative;  
    boundary="_002_4E36B98B966D7448815A3216ACF82AA201ED633ED1MBX3THNDRBIRD_"  
MIME-Version: 1.0  
--_002_4E36B98B966D7448815A3216ACF82AA201ED633ED1MBX3THNDRBIRD_  
Content-Type: text/html; charset="utf-8"  
Content-Transfer-Encoding: base64
```

```
SWYgeW91IGFyZSByZWFKaW5nIHRoaXMsIHlvdSBwcm9iYWJseSBzaG91bGQganVzdCBza2lwIHRoaX  
MgY2hhcHRlciBhbmQgZ28gdG8gdGhlIG5leHQgb25lLiBEbyB5b3UgcmlVhbGx5IGhhdmUgdGhlIHRp  
bWUgdG8gdHlwZSB0aGlzIHdob2xliHN0cmZyBpbj8gWW91IGFyZSBvYnZpb3VzbHkgdGFsZW50ZW  
QuIE1heWJlIHlvdSBzaG91bGQgY29udGFjdCB0aGUgYXV0aG9ycyBhbmQgc2VlIGlmIH
```

# Base64

```
GET /X29tbVEuYC8=/index.htm
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: www.practicalmalwareanalysis.com
Connection: Keep-Alive
Cookie: Ym90NTQxNjQ
```

```
GET /c2UsYi1kYWM0cnUjdFlvbiAjb21wbFU0YP==/index.htm
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: www.practicalmalwareanalysis.com
Connection: Keep-Alive
Cookie: Ym90NTQxNjQ
```

**URL và Cookie được mã hóa bằng Base64**

# Giải mã các URLs

❑ Tùy chỉnh “indexing string:

**aABCDEFGHIJKLMNOPQRSTUVWXYZbcdefghijklmnopq**

**rstuvwxyz0123456789+/-**

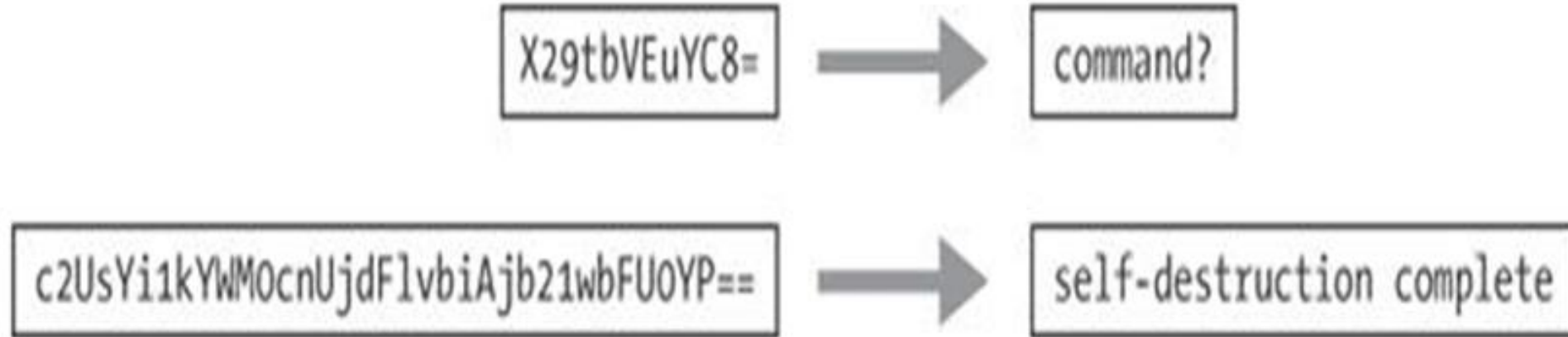
❑ Tìm ký tự = ở chuỗi đã được encoding

X29tbVEuYC8= → \_ommQ.` /

c2UsYi1kYWM0cnUjdFlvbiAjb21wbFU0YP== → se,b-dac4ru#tYon #omplU4`

*Unsuccessful attempt to decode Base64 string due to nonstandard indexing string*

# Giải mã các URLs



*Successful decoding of Base64 string using custom indexing string*



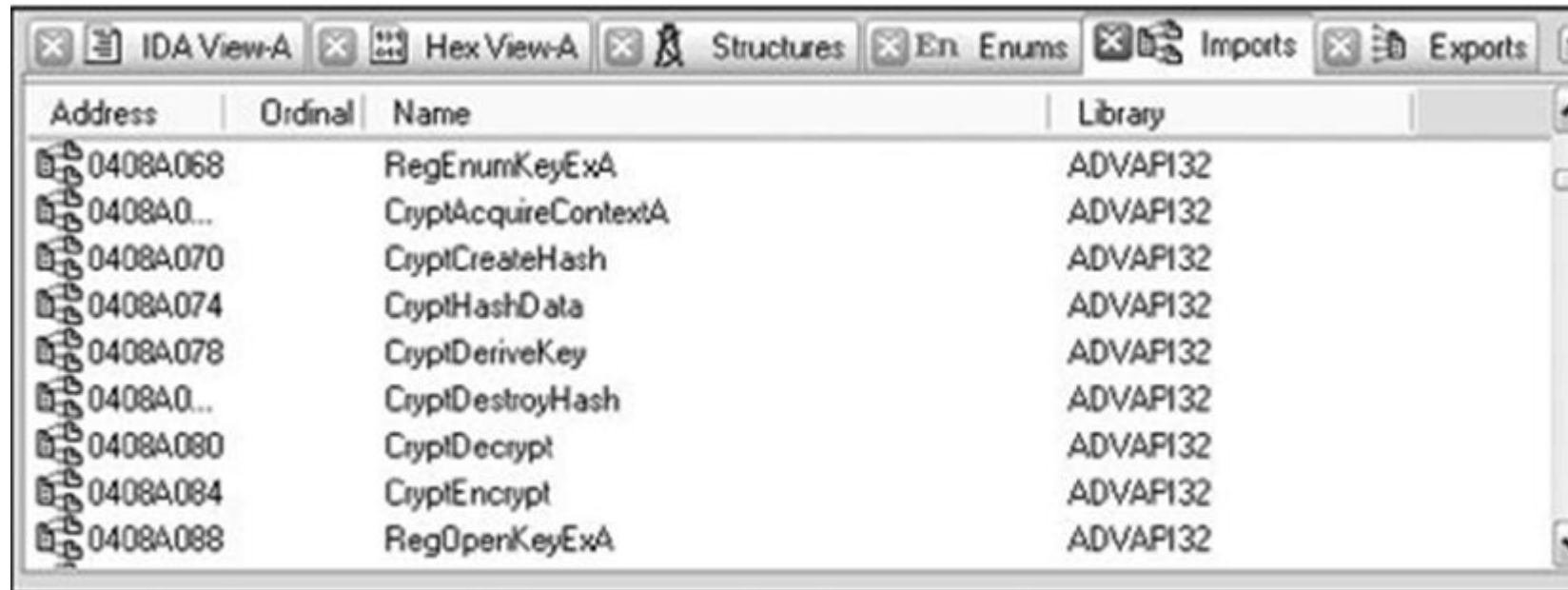
# Mã hóa

- ❑ Thuật toán mã hóa đơn giản
- ❑ Thuật toán mã hóa mạnh
- ❑ Thuật toán mã hóa tùy chỉnh

# Các thuật toán mã hóa mạnh

- ❑ AES, RSA,
- ❑ Chống lại các tấn công brute-force,
- ❑ Các thư viện mật mã chuẩn dễ bị phát hiện (các hàm được import, function matching, các hằng số mật mã),
- ❑ Yêu cầu nhiều tài nguyên tính toán

# Các thuật toán mã hóa mạnh



The screenshot shows the 'Imports' window in IDA Pro. The window title bar includes tabs for 'IDA View-A', 'Hex View-A', 'Structures', 'En Enums', 'Imports', and 'Exports'. The 'Imports' tab is active, displaying a list of imported functions from the ADVAPI32 library. The list is organized into four columns: Address, Ordinal, Name, and Library. The functions listed are cryptographic in nature, including RegEnumKeyExA, CryptAcquireContextA, CryptCreateHash, CryptHashData, CryptDeriveKey, CryptDestroyHash, CryptDecrypt, CryptEncrypt, and RegOpenKeyExA.

Address	Ordinal	Name	Library
0408A068		RegEnumKeyExA	ADVAPI32
0408A0...		CryptAcquireContextA	ADVAPI32
0408A070		CryptCreateHash	ADVAPI32
0408A074		CryptHashData	ADVAPI32
0408A078		CryptDeriveKey	ADVAPI32
0408A0...		CryptDestroyHash	ADVAPI32
0408A080		CryptDecrypt	ADVAPI32
0408A084		CryptEncrypt	ADVAPI32
0408A088		RegOpenKeyExA	ADVAPI32

*IDA Pro imports listing showing cryptographic functions*

# Mã hóa

- ❑ Thuật toán mã hóa đơn giản
- ❑ Thuật toán mã hóa mạnh
- ❑ Thuật toán mã hóa tùy chỉnh

# Thuật toán mã hóa tùy chỉnh

- ❑ Do người dùng tự phát triển
- ❑ Tùy chỉnh của các thuật toán mã hóa được công bố
- ❑ Ví dụ: Một vòng XOR, sau đó Base64
- ❑ Gây khó khăn cho quá trình phân tích

# Giải mã

- ❑ Lập trình các hàm giải mã
- ❑ Có thể sử dụng lại những hàm trong chính mã độc

# Lập trình các hàm giải mã

## ❑ Một số hàm chuẩn sẵn có

*Sample Python Base64 script*

```
import string
import base64

example_string = 'VGhpcyBpcyBhIHRlc3Qgc3RyaW5n'
print base64.decodestring(example_string)
```

*Sample Python NULL-preserving XOR script*

```
def null_preserving_xor(input_char, key_char):
    if (input_char == key_char or input_char == chr(0x00)):
        return input_char
    else:
        return chr(ord(input_char) ^ ord(key_char))
```

# Lập trình các hàm giải mã

## ❑ Một số hàm chuẩn sẵn có

*Sample Python custom Base64 script*

```
import string
import base64

s = ""
custom = "9ZABCDEFGHIJKLMNPOQRSTUVWXYZabcdefghijklmnopqrstuvwxyz012345678+/"
Base64 = "ABCDEFGHIJKLMNPOQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

ciphertext = 'TEgobxZobxZgGFPkb20='

for ch in ciphertext:
    if (ch in Base64):
        s = s + Base64[string.find(custom, str(ch))]
    elif (ch == '='):
        s += '='

result = base64.decodestring(s)
```



# Lập trình các hàm giải mã

## ❑ Một số thư viện mật mã (PyCrypto,...)

*Sample Python DES script*

```
from Crypto.Cipher import DES
import sys

obj = DES.new('password',DES.MODE_ECB)
cfile = open('encrypted_file','r')
cbuf = f.read()
print obj.decrypt(cbuf)
```

# Nội dung

1. Nén

2. Mã hóa

3. Làm rối mã

4. Một số cơ chế khác

# Làm rối mã

- ❑ Mục tiêu: Ngăn chặn việc phân tích, phát hiện dựa vào dấu hiệu (signature) và chống dịch ngược
- ❑ Code được Obfuscation và mutation
- ❑ Phát hiện debuggers và máy ảo nó sẽ dừng và không thực thi hành vi của nó nữa

# Obfuscation

- ❑ Đặt lại các lệnh, điều kiện bị đảo ngược, tên các thanh ghi khác nhau, trật tự khác nhau,...
- ❑ Các lệnh JUMP và NOP được chèn vào những vị trí ngẫu nhiên
- ❑ Bộ Garbage opcodes được chèn vào các khu vực không thể chèn mã
- ❑ Các lệnh được thay thế bằng những lệnh khác nhưng vẫn đảm bảo giống chức năng

# Polymorphic Viruses

- ❑ Mỗi bản sao tạo ra một bản mã ngẫu nhiên mới của cùng một virus
- ❑ Mã hóa > giải mã > Mã hóa > ...

# Metamorphic Viruses

- ❑ Lai tạo và kết hợp nhiều kiểu đa hình khác nhau trong cùng một virus
- ❑ Tự động biến đổi, lai tạp, hình thành các thể hệ virus  $F_1, F_2, F_3 \dots F_n$
- ❑ Các đoạn mã độc được rải rác và nằm ở nhiều nơi

# Zmist

❑ “Islands” của mã được tích hợp vào các vị trí ngẫu nhiên trong chương trình của máy lưu trữ và liên kết bằng các lệnh nhảy

❑ Virus tự ghép các phần của nó (tích hợp mã)

```
Instruction 4 ←  
Instruction 5 ←  
jmp  
garbage  
start:  
Instruction 1 ←  
Instruction 2 ←  
jmp  
garbage  
Instruction 3 ←  
jmp  
garbage
```

→

```
Instruction 2 ←  
jmp  
garbage  
Instruction 3 ←  
jmp  
garbage  
Instruction 5 ←  
jmp  
start:  
Instruction 1 ←  
jmp  
Instruction 4 ←  
jmp
```

→

```
Instruction 3 ←  
Instruction 4 ←  
jmp  
garbage  
Instruction 5 ←  
jmp  
start:  
Instruction 1 ←  
jmp  
garbage  
Instruction 2 ←  
jmp  
garbage
```

# Nội dung

**1. Nén**

**2. Mã hóa**

**3. Làm rối mã**

**4. Một số cơ chế khác**



# Một số cơ chế khác

- ❑ **Anti debugging**  
**(PMA chapter 16)**
- ❑ **Anti VM**  
**(PMA chapter 17)**

# Nội dung

- 1. Nén**
- 2. Mã hóa**
- 3. Làm rối mã**
- 4. Một số cơ chế khác**