BÁO CÁO BÀI TẬP

Môn học: Kỹ thuật phân tích mã độc

Tên chủ đề: BÀI TẬP NHÓM SỐ 02

GVHD: ThS Nguyễn Công Danh

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT137.Q11.ANTT

STT	Họ và tên	MSSV	Email
1	Nguyễn Thanh Hưng	22520517	22520517@gm.uit.edu.vn
2	Từ Chí Kiên	22520713	22520713@gm.uit.edu.vn
3	Hà Minh Quân	22521177	22521177@gm.uit.edu.vn
4	Nguyễn Nhật Quang	22521203	22521203@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá
1	Xác thực tệp PE	100%
2	Compression và encoding	100%
3	Tạo file .bin chứa dữ liệu packed và các hàm	100%
4	Tạo stub unpack và extract	100%
5	Kết quả Virus Total và chạy chương trình	100%

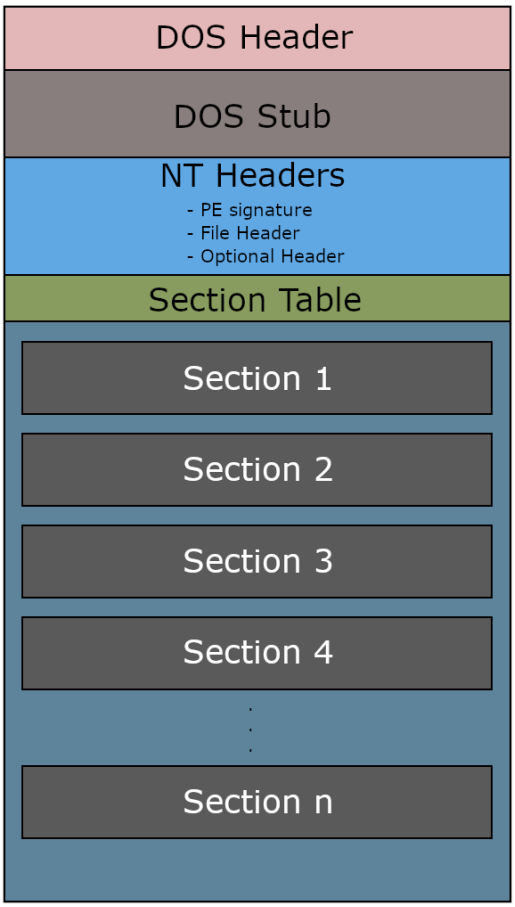
Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

[1] Ghi nội dung công việc

BÁO CÁO CHI TIẾT

1. Xác thực tệp PE

Cấu trúc file PE:



Bước đầu tiên phải xác định chương trình được pack/unpack phải kiểm tra tính hợp lệ của tệp PE.

Xác định thông qua 2 thành phần chính:

- Chuỗi "MZ" nằm trong DOS header

- Chữ ký "PE00" nằm trong NT header

Trong ngôn ngữ c:

Đối với chuỗi "MZ" có thể sử dụng IMAGE_DOS_HEADER trong thư viện windows.h để lấy:

```
// Get DOS header info
fread(&dosheader, sizeof(dosheader), 1, fptr);
// Check if the file is a PE file via "MZ" signature
if(dosheader.e_magic != IMAGE_DOS_SIGNATURE){ // IMAGE_DOS_SIGNATURE = 0x5A4D = MZ
    printf("Can't find MZ in DOS header, not a PE file.\n");
    fclose(fptr);
    return 1;
}
```

Đối với chữ ký "PE00" bằng cách chuyển đến NT header sử dụng fseek

```
// Move to nt headers
fseek(fptr, dosheader.e_lfanew, SEEK_SET);
// Get PE signature
fread(&Signature, sizeof(DWORD), 1, fptr);
if(Signature != IMAGE_NT_SIGNATURE){ // IMAGE_NT_SIGNATURE = 0x00004550 = PE00
    printf("Can't find PE signature.\n");
    fclose(fptr);
    return 1;
}
```

2. Compression và Encoding

Tiếp theo cho compression Huffman Coding: Là một thuật toán dựa trên tần số(số lần xuất hiện) xuất hiện của dữ liệu, cụ thể ở đây là byte. Byte có tần số các cao thì số bit đại diện thay thế càng thấp.

Thứ tự thực hiện:

- Tính tần số của các byte

- Tạo cây Huffman bằng cách tạo từ byte có tần số nhỏ đến lớn.

Ví dụ A(5), B(1), C(4)

```
   A
  /  \
B      C
```

- Tạo code định dạng cho byte thay thế thường 0 cho trái và 1 cho phải. Nếu theo ví dụ trên thì A(0), B(00), C(01)

- Thay thế các byte thành định dạng bit của nó và ghi lại vào file với padding nếu không đủ chiều dài.

Sau đó dữ liệu sẽ được encode sử dụng thuật toán Caesar Cipher, là thuật toán thay mỗi ký tự trong văn bản gốc được thay thế bằng một ký tự khác cách nó một số vị trí cố định trong bảng chữ cái. Trong trường hợp này sẽ là byte từ 0x00 – 0xFF.

```
int caesar_encrypt_mem(const uint8_t *in, size_t in_size, uint8_t **out, size_t *out_size, int shift)
    if (!in || in_size == 0){ // Make sure input size not empty
        return -1;
    }
    uint8_t *obuf = malloc(in_size); // Allocate output buffer same size as input
    if (!obuf){
        return -1;
    }
    for (size_t i=0; i<in_size; i++){
        // shift forward input via shift variable.
        obuf[i] = (uint8_t)((in[i] + shift) & 0xFF); // Use 0xFF or mod 256 to not overflow
    }
    *out = obuf; // save output
    *out_size = in_size; // save output size
    return 0;
}
```

3. Tạo file .bin chứa dữ liệu packed và các hàm

Về phần này trong code sẽ kiểm tra file pe và lấy kích thước file

```
    // Check that the input is a valid PE file
    if(check_validPEfile(inpath)){
        printf("Not a valid PE file");
        return 1;
    }

    // Open the input file for reading in binary
    FILE *f = fopen(inpath, "rb");

    // Seek to end to measure file size
    fseek(f,0,SEEK_END);
    long sz = ftell(f);
    fseek(f,0,SEEK_SET);

    if(sz <= 0){
        // empty or invalid size
        fclose(f);
        fprintf(stderr,"Empty file\n");
        return 1;
    }
```

Sau đó đọc toàn bộ file và buffer

```
// Allocate buffer to read the entire file
uint8_t *buf = malloc(sz);
if(!buf){
    fclose(f);
    return 1;
}

// Read file into buf
if(fread(buf,1,sz,f) != (size_t)sz){
    perror("read");
    free(buf);
    fclose(f);
    return 1;
}
fclose(f);
```

Compress và encode tất cả dữ liệu đó

```
// Compress the buffer in-memory using Huffman
uint8_t *comp = NULL; size_t comp_sz = 0;
if(huff_compress_mem(buf, (size_t)sz, &comp, &comp_sz) != 0){
    fprintf(stderr,"compress failed\n");
    free(buf);
    return 1;
}
free(buf);

// Encrypt the compressed data with Caesar cipher (shift cipher)
uint8_t *enc = NULL; size_t enc_sz = 0;
if(caesar_encrypt_mem(comp, comp_sz, &enc, &enc_sz, shift) != 0){
    fprintf(stderr,"encrypt failed\n");
    free(comp);
    return 1;
}
free(comp);
```

Ghi vào file tên là blob.bin

```
// Write the encrypted blob out to a file
FILE *out = fopen(outpath, "wb");
if(!out){ perror("open out");
    free(enc);
    return 1;
}
if(fwrite(enc,1,enc_sz,out) != enc_sz){
    perror("write");
    free(enc);
    fclose(out);
    return 1;
}
```

4. Tạo stub unpack và extract

Trong chương trình này sẽ import code và dữ liệu pack từ file blob.o

```
extern const unsigned char _binary_blob_bin_start[];
extern const unsigned char _binary_blob_bin_end[];
```

Sau đó đi đến vị trí của blob.bin và chạy các hàm decompress và decode

```
// Pointer to embedded blob
const uint8_t *blob = _binary_blob_bin_start;
size_t blob_sz = (size_t)(_binary_blob_bin_end - _binary_blob_bin_start);
if(blob_sz == 0){
    fprintf(stderr,"No embedded blob\n");
    return 1;
}

// Decrypt in memory using Caesar cipher (shift=3)
uint8_t *dec = NULL; size_t dec_sz = 0;
if(caesar_decrypt_mem(blob, blob_sz, &dec, &dec_sz, 3) != 0){
    fprintf(stderr,"decryption failed\n");
    return 1;
}

 // Decompress using Huffman
uint8_t *out = NULL; size_t out_sz = 0;
if(huff_decompress_mem(dec, dec_sz, &out, &out_sz) != 0){
    fprintf(stderr,"decompress failed\n");
    free(dec);
    return 1;
}
free(dec);
```

Ghi vào một file unpacked.exe

```c
// write unpacked.exe to current directory
const char *outname = "unpacked.exe";
FILE *f = fopen(outname, "wb");
if(!f){
    perror("fopen");
    free(out);
    return 1;
}
if(fwrite(out, 1, out_sz, f) != out_sz){
    perror("fwrite");
    fclose(f);
    free(out);
    return 1;
}
fclose(f);
free(out);
```

Một hàm chạy chương trình vừa mới unpack sẽ chạy để thực thi chương trình

```c
static int launch_exe_and_wait(const char *path){
    STARTUPINFOA si; PROCESS_INFORMATION pi;
    ZeroMemory(&si, sizeof(si)); si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));
    // Build mutable command line string
    char cmd[MAX_PATH+1];
    snprintf(cmd, sizeof(cmd), "\"%s\"", path); // Use cmd to run the program

    BOOL ok = CreateProcessA(
        NULL, cmd, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi);
    if(!ok){
        fprintf(stderr, "CreateProcess failed: %lu\n", GetLastError());
        return -1;
    }
    WaitForSingleObject(pi.hProcess, INFINITE);
    CloseHandle(pi.hThread); CloseHandle(pi.hProcess);
    return 0;
}
```

5. Kết quả Virus Total và chạy chương trình

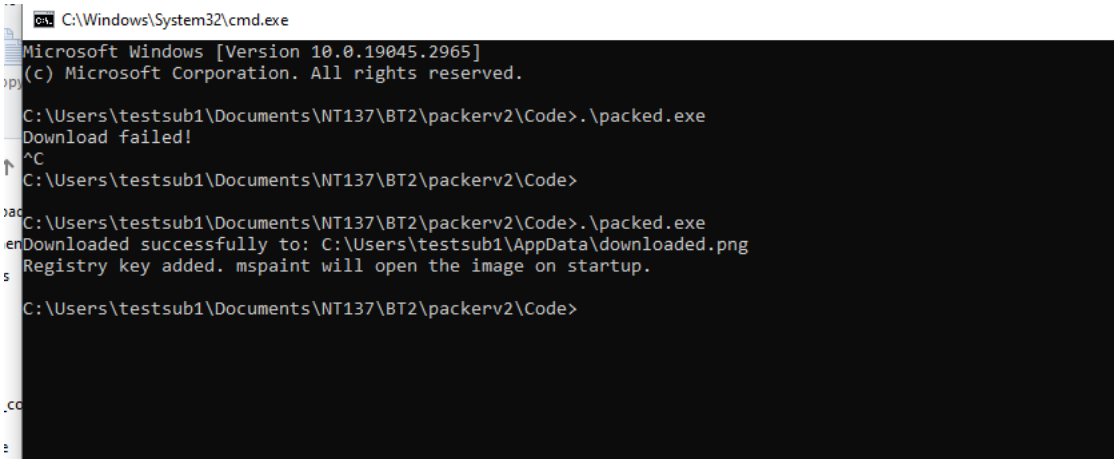
Để complie chương trình làm theo các bước như sau:

```
PS C:\Users\testsub1\Documents\NT137\BT2\packerv2\Code>  gcc -O2 -c huff_mem.c caesar.c file_util.c
PS C:\Users\testsub1\Documents\NT137\BT2\packerv2\Code>  gcc -O2 -o compressor.exe compressor.c huff_mem.o caesar.o file_util.o
PS C:\Users\testsub1\Documents\NT137\BT2\packerv2\Code> ./compressor.exe harmless_download.exe 3 blob.bin
Wrote blob blob.bin (7450785 bytes)
PS C:\Users\testsub1\Documents\NT137\BT2\packerv2\Code> objcopy -I binary -O pe-x86-64 -B i386:x86-64 blob.bin blob.o
PS C:\Users\testsub1\Documents\NT137\BT2\packerv2\Code> gcc -O2 -o packed.exe stub.c huff_mem.o caesar.o blob.o -luser32
```

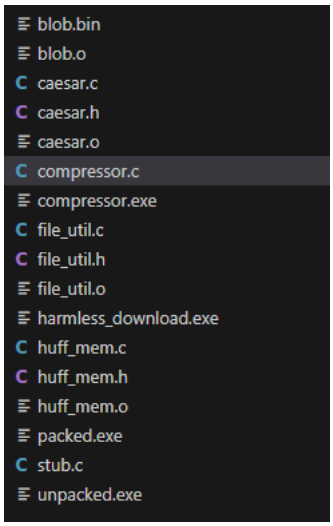
gcc -O2 -c huff_mem.c caesar.c file_util.c: Complie các file hỗ trợ packing trước
gcc -O2 -o compressor.exe compressor.c huff_mem.o caesar.o file_util.o: Sau đó complie chương trình tạo file .bin.

./compressor.exe harmless_download.exe 3 blob.bin: Tạo chương trình với chương trình muốn pack (harmless_download.exe) với 3 là số shift cho Caesar Cipher và kết quả ghi vào blob.bin
objcopy -I binary -O pe-x86-64 -B i386:x86-64 blob.bin blob.o: Ở đây gắn file .bin vào object và đồng thời chuyển sang 32bit.
gcc -O2 -o packed.exe stub.c huff_mem.o caesar.o blob.o -luser32: Complie để tạo chương trình được pack có stub unpack.

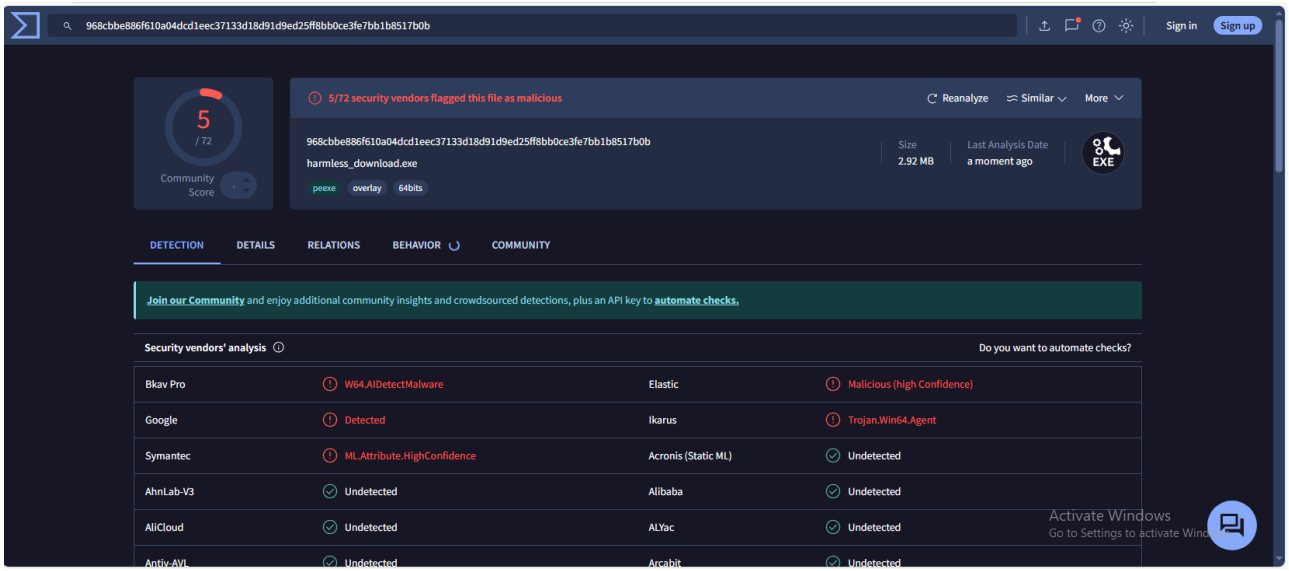
Kết quả khi chạy chương trình packed.exe, thực thi như chương trình harmless_download.exe



Nhưng do chương trình này unpack lưu vào file rồi mới chạy nên trên thư mục chứa chương trình sẽ chứa thêm file unpacked.exe



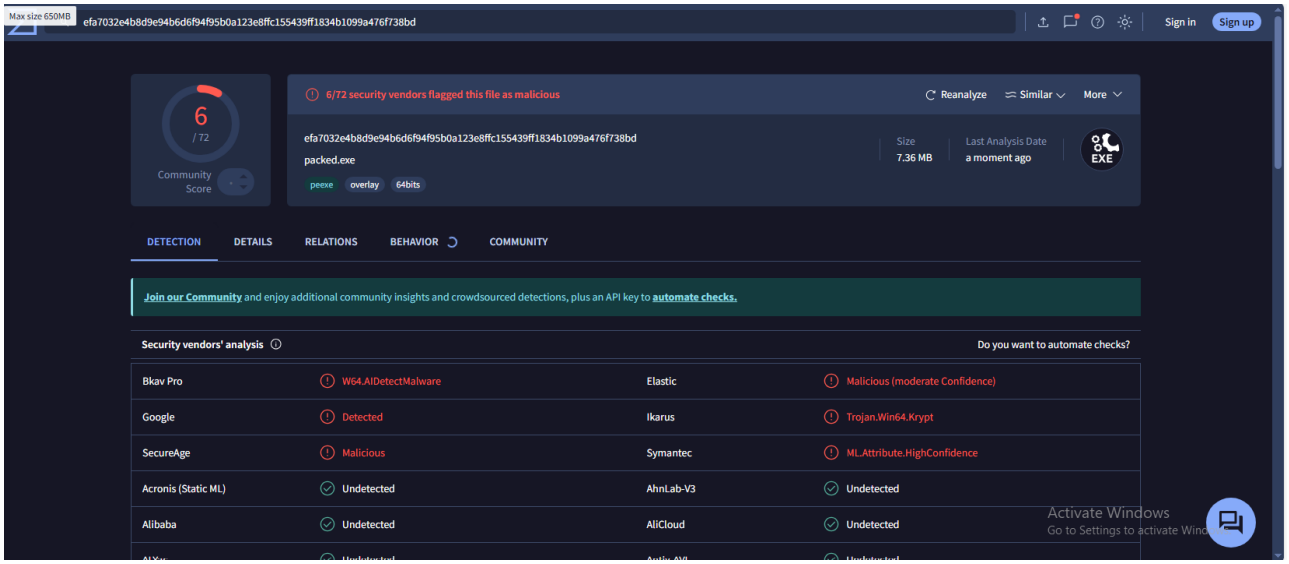
Kết quả của file ở BT1



Theo kết quả trên có 5 AV phát hiện được trong số 72
Bkav Pro W64.AIDetectMalware
Elastic Malicious (high Confidence)
Google Detected
Ikarus Trojan.Win64.Agent
Symantec ML.Attribute.HighConfidence

Và file có mã hash
968cbbe886f610a04dcd1eec37133d18d91d9ed25ff8bb0ce3fe7bb1b8517b0b

Còn với file packed.exe



Kết quả trên cho thấy có 6 AV phát hiện được:
Bkav Pro W64.AIDetectMalware
Elastic Malicious (moderate Confidence)

Google Detected
Ikarus Trojan.Win64.Krypt
SecureAge Malicious
Symantec ML.Attribute.HighConfidence

Với mã hash là :
efa7032e4b8d9e94b6d6f94f95b0a123e8ffc155439ff1834b1099a476f738bd

Ở đây có thể thấy là packer chưa hiệu quả, do nhiều AV phát hiện được file packed.exe hơn file mã độc chính thống.

6. Nguồn tham khảo

Cấu trúc PE file: https://0xrick.github.io/win-internals/pe2/

MZ và chữ ký PE file: https://kienmanowar.wordpress.com/2014/01/16/tim-hieu-pe-file-qua-cac-vi-du-co-ban/

Huffman algorithm: https://www.geeksforgeeks.org/c/huffman-coding-in-c/

Huffman compression: https://medium.com/stantmob/data-compression-with-huffman-coding-ad7bcb07c5d5

Caesar: https://www.geeksforgeeks.org/ethical-hacking/caesar-cipher-in-cryptography/

Tham khảo về các hoạt động của một chương trình tự động Extract:

https://cplusplus.com/forum/general/56128/

Nhúng các blob nhị phân bằng gcc mingw:
https://stackoverflow.com/questions/2627004/embedding-binary-blobs-using-gcc-mingw

---

HẾT