Rio de Janeiro, 10 de fevereiro de 2011
Revisão 4 em 12 de abril de 2011

O processo de construção de um analisador de executáveis

por Fernando Mercês (fernando@mentebinaria.com.br)

1. Introdução

Um analisador de executáveis é um software capaz de prover informações sobre um executável que podem ser muito úteis para pesquisadores de Segurança da Informação na análise de malware, forense computacional ou engenharia reversa.

Este artigo objetiva demonstrar como um analisador de executáveis é construído, abordando técnicas para uma análise precisa e eficiente. É utilizado como base, um software de código aberto chamado "pev", de desenvolvimento próprio, que analisa binários PE32 (usados no MS-Windows) e pode ser compilador tanto em sistemas UNIX-like quanto no próprio Windows, pois foi escrito em ANSI C.

O processo de construção exige conhecimento do ambiente e da linguagem de programação escolhida. O estudo necessário para tal é de grande valor na carreira do pesquisador de segurança.

2. O executável

Antes de iniciar, precisamos compreender o que é um arquivo executável. Sabemos que todo e qualquer arquivo no disco rígido não passa de uma sequência de bits armazenados por um processo elteromagnético nos pratos do disco. A diferença entre um arquivo MP3 e um PNG, por exemplo, é a forma como esses bits serão interpretados.

No caso do executável, os bits presentes no arquivo representam instruções de máquina (Assembly) para o microprocessador da arquitetura em questão (Intel, SPARC etc). Veja:

Binário	Decimal	Hexadecimal	Assembly x86	ASCII
01010101	85	55	push ebp	U

O mesmo byte (conjunto de 8 bits) pode ser interpretado de diversas formas. De fato, é por este motivo que um software editor hexadecimal abre qualquer tipo de arquivo, inclusive áreas do disco diretamente, lendo byte a byte sem qualquer interpretação.

O arquivo executável é um formato complexo (bem diferente de um arquivo em texto puro – clear text, por exemplo). Além dos bytes referentes ao código do programa em si, é preciso adicionar milhares de bytes que constituem informações para guiar o kernel do SO à execução do binário. É preciso informar, por exemplo, para qual arquitetura o executável foi compilado, quanto de memória será alocada para rodar o programa, que partes do programa em memória serão exclusivas, somente para leitura e mais uma série de diretivas.

3. O formato

Para suprir todas essas necessidades de informações é que existem os formatos. Estes definem padrões que um arquivo deve seguir para ser corretamente interpretado pelo seu programa associado ou pelo próprio SO, no caso de um executável.

Atualmente lidamos basicamente com dois formatos de executáveis: o PE e o ELF. O primeiro é utilizado pela família Windows e o segundo, pelos sistemas UNIX-like.

O que um analisador precisa informar?

Já dissemos que num executável **não** há somente o código que o programador escreveu na linguagem de programação convertido para código de máquina. Por isso, em tese, um analisador deveria nos dar toda esta informação "escondida" no executável. Os desenvolvedores dos formatos de executáveis geralmente liberam esta documentação porque todos os compiladores precisam gerar executáveis compatíveis e, por isso, têm de conhecer a especificação. Então o primeiro passo para se construir um analisador é obter a documentação do formato:

Formato PE
http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/pecoff_v8.docx

Formato ELF
http://www.mentebinaria.com.br/files/elf-spec.pdf

Na documentação do formato, constam todos os campos pré-definidos que se espera encontrar num executável. Mas é claro que nem tudo é necessário para se construir um bom analisador. Alguns campos possuem grande utilidade prática, enquanto outros raramente são necessários. Cabe a nós filtrar o que é importante para o objetivo.

4. O analisador

Um código que consiga interpretar os campos que o formato define num executável precisa:

- ✔ Verificar se o binário é de tal formato.
- ✔ Ler os bytes do binário de acordo com a especificação.
- ✔ Imprimir os nomes do campo e seus respectivos valores na tela.

Simples? Nem tanto. Geralmente um analisador é um software pequeno que roda rápido (porque já sabe o que vai ler), mas o código-fonte é grande e pode vir a ser complexo. Para um exemplo prático, imagine que o formato PE defina o seguinte:

- ➔ Para ser um arquivo PE válido, os primeiros dois bytes do arquivo binário devem ser 0x4D e 0x5A.

Neste caso, o analisador precisa fazer tal verificação:

```
int verify(char *filename)
{
    FILE *fp = fopen(filename, "rb");
    char bytes[2];

    fread(bytes, 2, 1, fp);

    if (bytes[0] == 'M' && bytes[1] == 'Z')
        return 1;

    return 0;
}
```

E fim de papo! Sim, um malware não pode alterar estes bytes, do contrário o Windows não o executará, portanto, não tenha medo em testar e encerrar o program caso não haja as letras MZ no início do arquivo, que são a representação em ASCII dos bytes 4D e 5A, em hexa. Experimente alterar um desses bytes de um executável PE e tente rodá-lo para ver o que acontece. Humm... será que um executável PE com o primeiro byte alterado passaria via e-mail pelo firewall de sua empresa?

Agora digamos que a especificação do formato PE também diga:

- ➔ 16 bytes à frente desta assinatura MZ encontra-se o *checksum* do arquivo, que tem um comprimento também de 2 bytes. Bastaria "andar" pelo arquivo para ler:

```
    unsigned short checksum;

    fseek(fp, 16, SEEK_CUR);
    fread(&checksum, 2, 1, fp);
    printf("%d\n", checksum);
```

PS.: Em C, o tipo short, que abrevia short int, tem 2 bytes na arquitetura Intel x86.

Seguindo essa lógica, podemos imprimir todos os campos de um binário, bastando apenas seguir a especificação do formato. No entanto, há recursos de linguagem que podem facilitar a vida. Veja um trecho interessante da biblioteca *windows.h* abaixo:

```
typedef struct _IMAGE_FILE_HEADER {
    WORD Machine;
    WORD NumberOfSections;
    DWORD TimeDateStamp;
    DWORD PointerToSymbolTable;
    DWORD NumberOfSymbols;
    WORD SizeOfOptionalHeader;
    WORD Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

No caso do C, se definirmos um tipo WORD e DWORD com o typedef, obtemos um struct prontinho para ser usado e com os nomes dos campos. O mesmo existe para o formato ELF. ;-)

5. Fazendo mais

Imprimir informações brutas faz parte, mas a graça de um analisador está em sua capacidade de fazer mais que isso. Por exemplo, tratar um *timestamp* ou realçar o Entry Point (EP) de um binário são técnicas simples e que vão ajudar muito quem utilizará o software.

6. Binários com proteção

Um bom analisador deve esperar um binário que contenha um packer, crypter ou qualquer outro tipo de proteção. Neste caso, é necessário estudar e entender a rotina da proteção, fazer engenharia reversa, e inserir rotinas no analyzer para detectar ou mesmo remover as proteções dos executáveis. Isso vai dar um trabalho extra (e constante, porque novas proteções não param de surgir, além de atualizações das proteções existentes) mas sua implementação depende do objetivo desejado. A maioria dos analisadores somente reconhece que há uma proteção (alguns dizem qual é ela, batendo a assinatura contra um banco de dados), mas poucos a removem sem plugins adicionais.

7. pev

Software livre (GPLv3) inicialmente desenvolvido para exibir o valor do campo “Product Version” de um executável PE.

```
fernando@brussels:~$ pev -c ~/winapp/wrar393br.exe
COFF header:
 Machine:                     0x14c
 Number of sections:          5
 Date/time stamp:             1268634487 (03/15/2010 at
06:28:07 AM)
 Symbol Table offset:         0
 Number of symbols:           0
 Size of optional header:     0xe0
 Characteristics:             0x103
```

Página do projeto: http://sourceforge.net/projects/pev

8. Conclusão

Conhecer bem os executáveis é obrigação de quem trabalha ou pretende trabalhar com análise de malware ou forense computacional e nada melhor que um estudo dirigido, que force resultados para atingir este objetivo. Desenvolver uma aplicação que interprete um executável “de cabo-a-rabo” é um ótimo começo. $