Artigo

Windows Portable Executable

Conhecendo por dentro nossos programas, neste artigo de Daniel Dummer.

Windows Portable Executable

Conhecendo por dentro nossos programas

Este artigo descreve o formato Portable Executable (PE), formato padrão para arquivos objetos (bpl, dpl, cpl, ocx, etc.) e executáveis (exe, dll, sys, scr, etc.) da plataforma Microsoft Windows, demonstrando suas seções, analisando sua estrutura e funcionamento.

Introdução

Este artigo descreverá o formato Windows Portable Executable (PE), formato muito comum tratando de formatos de arquivos executáveis e arquivos-objeto na plataforma Microsoft Windows.

O formato PE é basicamente uma estrutura padrão de armazenamento de dados de arquivos onde estão encapsuladas todas as informações necessárias (código, dados inicializados, dados não inicializados, etc.) ao sistema operacional (system loader) para sua leitura e execução.

Costumeiramente pouco estudado por desenvolvedores, devido até pela não obrigatoriedade de conhecimento específico e aprofundado no desenvolvimento de aplicativos, este assunto dificilmente é tópico de discussão ou debate na comunidade, porém este estudo e análise pode agregar em muito àqueles que buscam conhecer o mecanismo interno a fins de otimização, conhecimento em geral e, principalmente, proteção de software.

No caso de proteção de software, é de extrema importância seu conhecimento, pois é nele que se encontram as informações básicas e fundamentais para funcionamento de um programa. Assim, é preciso conhecê-lo detalhada e aprofundadamente a fim de criar métodos, armadilhas e bloqueios para prevenção de descarregadores de memória (memory dumpers), debuggers e outros tipos de utensílios e ferramentas de cracking[1].

História

O formato Portable Executable foi desenvolvido pela Microsoft e padronizado em 1993 pelo Comitê de Padrões de Interfaces de Ferramentas (Tool Interface Standard Committee), formado pela Microsoft, Intel, Borland, Watcom, IBM, entre outras.

O modelo PE foi desenvolvido com base no modelo COOF[2], isto porque a maioria de seus criados foram os mesmos que desenvolveram e codificaram o COOF. Com isso aproveitaram muito do código deste modelo, já que o mesmo funcionava muito bem e já havia sido testado exaustivamente. Sendo assim, seria apenas necessário fazer as adaptações necessárias e exigidas pelas novas plataformas que usufruírem deste formato.

O nome Portable é devido realmente a sua portabilidade, que possibilita sua implementação em diversas plataformas (x86, MIPS®, Alpha, entre outros) sem que sejam necessárias alterações em seu formato. É lógico que diversas alterações (como codificação binária de instruções de CPU, etc.) são necessárias para funcionamento nestas plataformas, porém o detalhe interessante é que não foi necessário reescrever do zero carregadores de sistemas para memória (system loader) do sistema operacional e outras ferramentas para desenvolvimento.

O formato PE possui as seguintes estruturas de dados: cabeçalho MZ DOS, fragmento (stub) DOS, cabeçalho de arquivo PE, cabeçalho de imagem opcional, tabela de seções (que possui uma lista de cabeçalhos de seção), diretórios de dados (que contém os ponteiros para as seções) e ultimamente as seções propriamente ditas, conforme **Figura 1.**

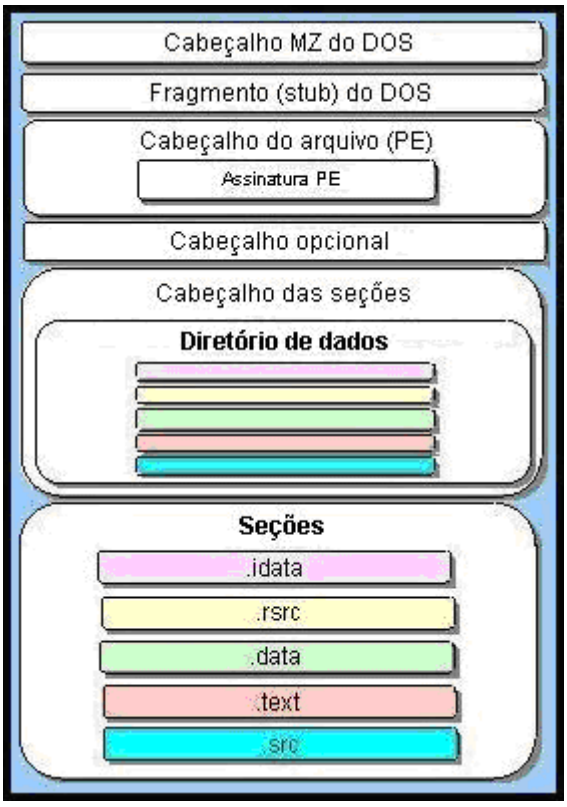

Figura 1. Estrutura do formato PE

Os arquivos PE, quando carregados na memória, são bastante similares aos arquivos no disco. Com isto, o carregador (system loader) pode executar essa operação mais rapidamente, apenas tendo que mapear os endereços do arquivo para endereços de memória.

Agora será analisado os dados de cada seção existente num arquivo PE.

Cabeçalho DOS

Os primeiros bytes de qualquer arquivo PE constituem o cabeçalho DOS. É assim com todo e qualquer arquivo executável ou arquivo-objeto PE. Além disso, os primeiros dois bytes deste cabeçalho serão a assinatura deste formato, sempre formada pelos bytes "MZ" (4D 5A em

hexadecimal), ou seja, para um arquivo ser do formato PE, o mesmo deve obrigatoriamente começar por essa seqüência.

O cabeçalho DOS é constituído de 64 bytes, dispostos da seguinte maneira:

1. Assinatura “MZ” (2 bytes)
2. Tamanho da última página (2 bytes)
3. Total de páginas (2 bytes)
4. Itens de relocação (2 bytes)
5. Tamanho do cabeçalho DOS (2 bytes) – Estes bytes indicam o quantidade seqüências de 16 bytes (0F em hexadecimal) contidas no cabeçalho, ou seja, para determinar o tamanho do cabeçalho basta multiplicar o valor encontrado neste campo por 16
6. Tamanho mínimo da memória (2 bytes) – sempre encontrado “00 00” em hexadecimal
7. Tamanho máximo da memória (2 bytes) – sempre encontrado “FF FF” em hexadecimal
8. Valor inicial do registrador SS (Stack Segment) (2 bytes)
9. Valor inicial do registrador SP (Stack Pointer) (2 bytes)
10. Checksum do cabeçalho DOS (2 bytes)
11. Valor inicial do registrador IP (Instruction Pointer) (2 bytes)
12. Valor inicial do registrador CS (Code Segment) (2 bytes)
13. Offset do fragmento (stub) DOS (2 bytes)
14. Overlay (2 bytes)
15. Identificador OEM (2 bytes)

16. Informações OEM (2 bytes)

17. Bytes reservados (24 bytes)

18. Betov's CheckSum (4 bytes)

19. Offset do cabeçalho de arquivo PE (4 bytes)

Para melhor exemplicar, será usado, como base de dados para as figuras, o arquivo notepad.exe do Windows XP SP 2.

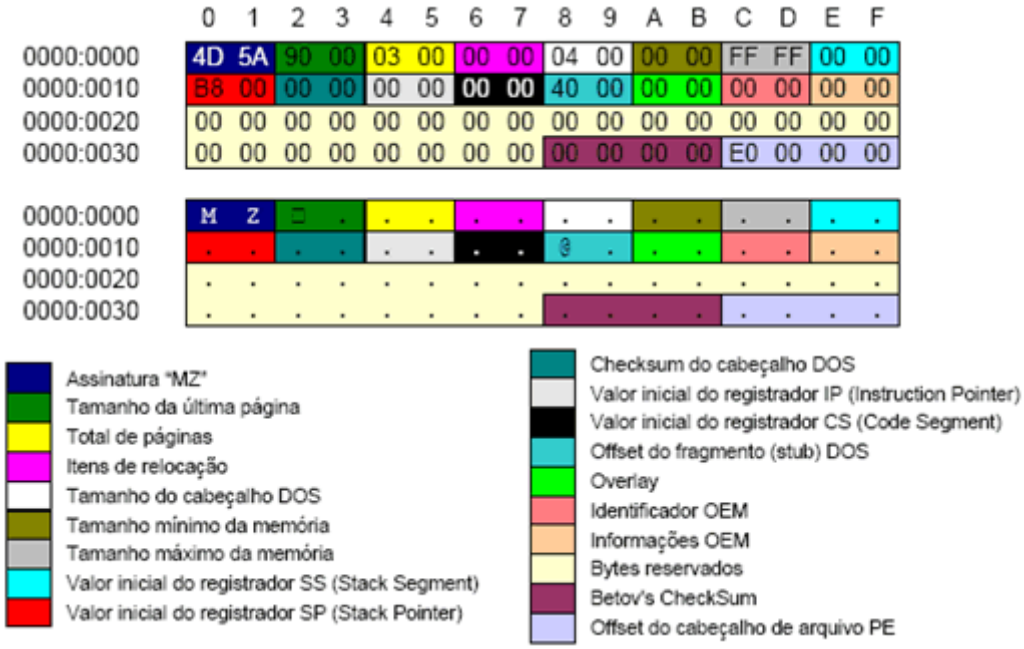

Figura 2. Cabeçalho DOS

Fragmento (Stub) do DOS

Como detalhado anteriormente no cabeçalho DOS, é encontrado no offset 0x18 (item 13) o fragmento (stub) do DOS.

Este fragmento na verdade é um executável embutido no cabeçalho PE, que é chamado caso o arquivo PE não possa ser executado.

Consiste num número muito pequeno de bytes (até por que o tamanho padrão do fragmento DOS é de 64 bytes), divididos em instruções de máquina e num texto. Assim, ao carregar o programa na memória, verifica o sistema é compatível, e caso não for, utiliza esta seção para exibir a mensagem de erro.

Cabeçalho do Arquivo

Como detalhado anteriormente no cabeçalho DOS, é encontrado no offset 0x3C (item 19) o cabeçalho do arquivo.

Os componentes do cabeçalho do arquivo são os seguintes:

1. Assinatura do cabeçalho PE (4 bytes) – sempre encontrado a seqüência "PE00" ("50 45 00 00" em hexadecimal)
2. Tipo de máquina previsto para rodar o executável (2 bytes)
3. Número de seções após o cabeçalho (2 bytes)
4. Carimbo TimeDateStamp - Data e hora de criação do arquivo (4 bytes)
5. Ponteiro para Tabela de Símbolos (4 bytes)
6. Número de Símbolos (4 bytes)
7. Tamanho do Cabeçalho Opcional (2 bytes)
8. Características do arquivo (definidas por bit flags) (2 bytes)

Figura 3. Cabeçalho do arquivo

Cabeçalho Opcional

Imediatamente após o cabeçalho do arquivo vem o cabeçalho opcional que, apesar do nome, está sempre presente. É que o COOF utiliza um cabeçalho para bibliotecas, mas não para objetos, que é chamado de opcional. Este cabeçalho indica mais alguns detalhes de como o binário deve ser carregado: o endereço inicial, a quantidade reservada para a pilha (stack), o tamanho do segmento de dados etc. Este cabeçalho contém informações de como o arquivo PE deve ser tratado.

Os componentes do cabeçalho opcional são os seguintes:

1. Valor 'Magic' (2 bytes) – define o tipo de arquivo (010B=Executável, 0107=Imagem ROM)
2. Maior versão do lincador[3] (linker) (1 byte)
3. Menor versão do lincador3 (linker) (1 byte)
4. Tamanho do código executável (4 bytes)
5. Tamanho de dados de inicialização (segmento de dados) (4 bytes)
6. Tamanho de dados de não-inicialização (segmento BSS) (4 bytes)
7. RVA[4] de entrada do código do executável (4 bytes)
8. RVA da base do código (4 bytes)

9. RVA da base dos dados (4 bytes)

10. Base da imagem - endereço de mapeamento preferencial (4 bytes)

11. Alinhamento da seção na RAM (4 bytes)

12. Alinhamento do arquivo em disco (4 bytes)

13. Versão máxima (2 bytes) do sistema operacional esperado[5]

14. Versão mínima (2 bytes) do sistema operacional esperado5

15. Versão máxima (2 bytes) do arquivo PE[6]

16. Versão mínima (2 bytes) do arquivo PE6

17. Versão máxima (2 bytes) do subsistema esperado[7]

18. Versão mínima (2 bytes) do subsistema esperado7

19. Versão do Win32 (4 bytes) – sempre zerado

20. Tamanho da imagem (4 bytes) – é a soma dos tamanhos dos cabeçalhos e seções. Este campo serve como dica para o carregador do sistema (system loader) saber quanto de memória alocar para carregar o programa

21. Tamanho dos cabeçalhos (4 bytes) - é a soma dos tamanhos dos cabeçalhos, incluindo diretórios de dados e cabeçalhos de seções

22. CheckSum do arquivo PE (4 bytes)

23. Subsistema requerido (2 bytes) – (normalmente possui o valor 2, referente a programas Win32 GUI)

24. Características de DLL (2 bytes)

25. Tamanho de reserva de pilha (4 bytes)

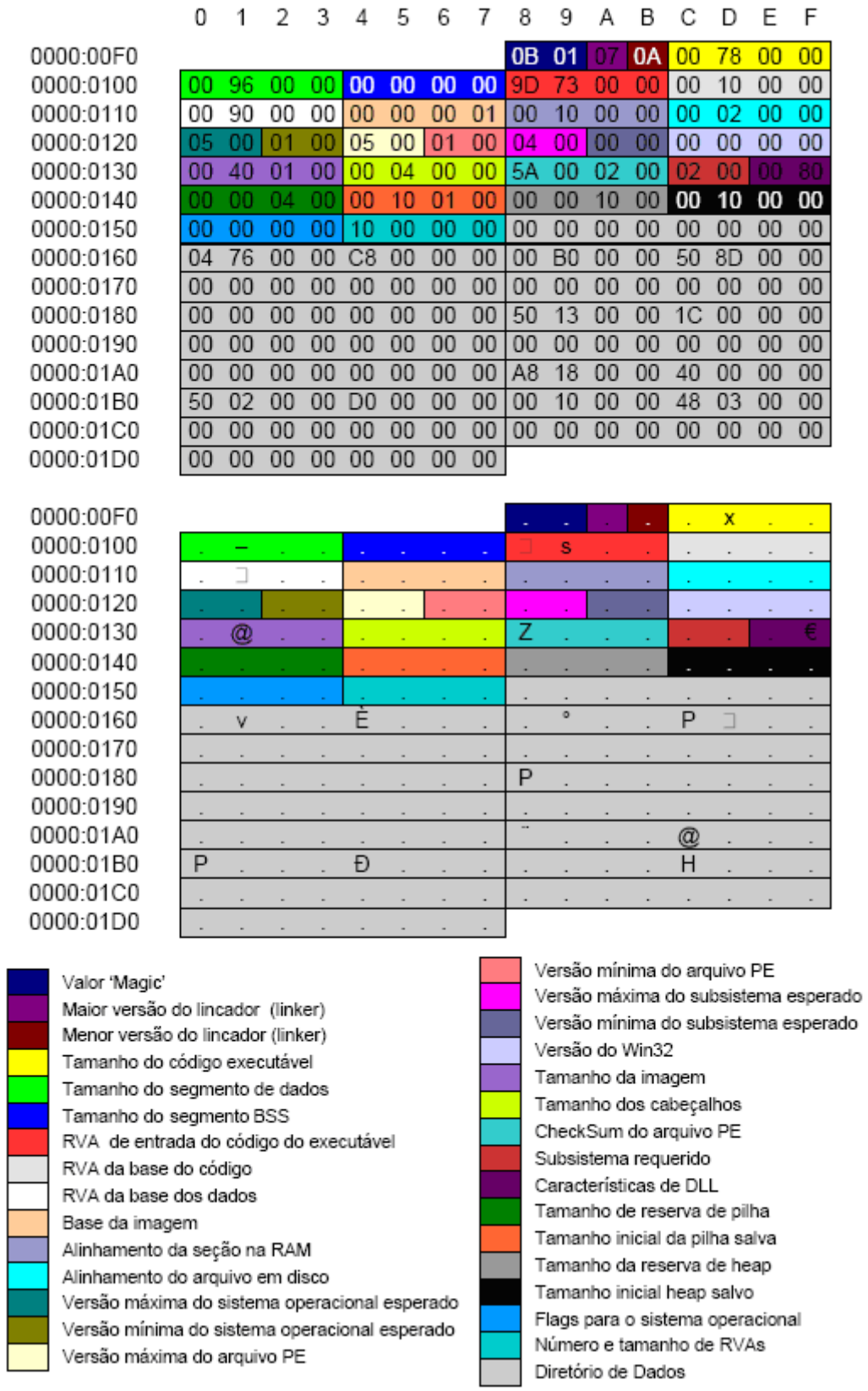

Figura 4. Cabeçalho opcional

26. amanho inicial da pilha salva (4 bytes)

27. Tamanho da reserva de heap (4 bytes)

28. Tamanho inicial heap salvo (4 bytes).

29. Flags para o carregador do sistema operacional (4 bytes)

30. Número e tamanho de RVAs (4 bytes)

31. Diretório de Dados (é um array de 16 descritores de diretórios, com localização (RVA) e o tamanho de cada peça de informação)

Cabeçalho de Seções

Os cabeçalhos de seção são as descrições que antecedem a seção propriamente dita.

Um cabeçalho de seção contém:

1. Um array (8 bytes) com o nome das seção

2. Tamanho virtual da seção (4 bytes)

3. Endereço físico da seção (4 bytes)

4. Tamanho alinhado (4 bytes) - tamanho dos dados da seção arredondado para cima para o próximo múltiplo do alinhamento de arquivo

5. Offset do início do arquivo em disco até os dados da seção (4 bytes)

6. Ponteiro para remanejamento (4 bytes) – apenas para arquivos-objeto

7. Ponteiro para números de linha (4 bytes) – apenas para arquivos-objeto

8. Número de remanejamentos (2 bytes) – apenas para arquivos-objeto

9. Quantidade de números de linha (2 bytes) – apenas para arquivos-objeto

10. Características que descrevem como a memória da seção deve ser tratada (4 bytes) (por bit flag)

Seções

Após os cabeçalhos das seções seguem as seções.

Existem vários tipos de seções, dependendo do seu conteúdo, e são nelas que ficam salvas as instruções, recursos (resources) e todos os dados e informações do programa propriamente dito. Cada seção possui algumas flags sobre alinhamento, o tipo de dados que contém, se pode ser compartilhada, etc.

Em resumo, é nesta seção onde são gravadas os códigos do programa.

Conclusões

Um bom conhecimento e compreensão de como é e funciona o formato PE leva a um bom conhecimento e compreensão do sistema operacional em um todo.

Conhecer como funcionam internamente suas bibliotecas e executáveis faz com que o desenvolvedor não saiba somente mais sobre programação, mas sim aprende e descobre tudo o que ocorre com sua aplicação e toda interação com o sistema operacional, ponto este fundamental para o desenvolvimento de software com qualidade e profissionalismo.

Daniel Dummer (danieldummer@gmail.com) – é Bacharelando em Ciência da Computação pelo Centro Universitário Feevale. Programador Delphi a 5 anos, trabalha com dbExpress em projetos cliente/servidor, além de projetos COM+ e ASP.

[1] Cracking é o nome dado a ações de modificações no funcionamento de um sistema, de maneira geralmente ilegal, para que determinados usuários

ganhem algo com isso.

[2] Common Object File Format: Formato comum de arquivo-objeto dos sistemas UNIX, VMS e VAX.

[3] A junção dos bytes dos itens 2 e 3 formam a versão do lincador utilizada para lincar o arquivo.

[4] É um endereço virtual relativo, utilizado para definir um endereço de memória caso se desconheça o endereço base. É o valor que, adicionado ao endereço base, fornece o endereço linear

[5] A junção dos bytes dos itens 13 e 14 formam a versão mínima esperada do sistema operacional

[6] A junção dos bytes dos itens 15 e 16 formam a versão do arquivo PE

[7] A junção dos bytes dos itens 17 e 18 formam a versão mínima esperada do subsistema

Por **Equipe**
Em 2005

Suporte ao aluno - Deixe a sua dúvida.

Poste aqui sua dúvida ou comentário que nossa equipe responderá o mais rápido possível.