PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
ENGENHARIA DE COMPUTAÇÃO

VÍRUS V2B

TRABALHO PRÁTICO 2

Autores: Bruna Silva, Bruno Porcher e Vinícius Fochi
Professor: Eduardo Bezerra
Disciplina: Arquitetura de Computadores I

PORTO ALEGRE, 04 DE JULHO DE 2008

OBJETIVOS

Desenvolver um programa em assembly para a arquitetura IA32, para execução no Linux (NASM) que ao ser executado realize alterações em um programa executável existente no mesmo diretório. O programa deverá apresentar as seguintes funcionalidades:

- Abertura de arquivos visando encontrar programa executável a ser alterado;
- Busca por determinadas instruções “alteráveis” em código de máquina no programa alvo executável;
- Ao encontrar uma instrução (ou trecho de instruções), realizar a alteração no programa alvo executável de forma a inserir o pseudo-virus (código malicioso) nesse arquivo executável;
- Gravar uma assinatura do pseudo-virus nesse arquivo;
- Gravar no disco o programa alvo executável devidamente alterado.

SOLUÇÃO PROPOSTA

Desenvolvemos um programa em assembly que verifica em todos os arquivos executáveis, do mesmo diretório em que se encontra o vírus, se tem a instrução especificada e caso a encontre o programa substitui a instrução por um código malicioso que faz o arquivo entrar em loop exibindo na tela uma mensagem pré-definida repetidamente e simultaneamente alocando espaço na pilha ocasionando "estouro de pilha" e então é exibida a mensagem "falha de segmentação". Essa alteração é feita no primeiro executável que encontrarmos a instrução, depois o programa encerra a verificação nos demais arquivos do diretório.

DESENVOLVIMENTO DO VIRUS

Instruções procuradas:

Instruções ADD procuradas	
Código em Hexadecimal	**Código em Assembly**
0x81C1	ADD ecx, [imediato]
0x81C2	ADD edx, [imediato]
0x81C3	ADD ebx, [imediato]

Código malicioso:

Instrução substituta	
Código em Hexadecimal	**Código em Assembly**

0x68,0x20,0x48,0x41,0x00,
0xba,0x04,0x00,0x00,0x00,
0x89,0xe1,0xbb,0x01,0x00,
0x00,0x00,0xb8,0x04,0x00,
0x00,0x00,0xcd,0x80,0x50,
0xeb,0xe5
	push 0x414820
	mov edx,4
	mov ecx,esp
	mov ebx,1
	mov eax,4
	int 0x80
	push eax
	jmp main

PASSO A PASSO:

1- Listar todos os arquivos contidos no seu diretório atual

Para realizar essa tarefa, utilizamos a syscall SYS_GETDENTS (equ 141) que lista todos os arquivos do diretório. Inicialmente enfrentamos alguns problemas, tais como:

- A função retorna algumas informações desnecessárias para o nosso programa, pois queríamos apenas o nome do arquivo.

Solução: para obter apenas o nome do arquivo, no inicio da execução o ECX aponta 10 bytes a frente do número obtido com a sys_getdents, armazenado em EDI.

- A listagem de arquivos exibe o diretório atual, o diretório pai **(“.” e “..”)** e ainda o nosso próprio vírus. O que causaria falha de segmentação ao tentar abrir alguns destes arquivos.

Solução: criamos alguns loops de testes que verificam se o arquivo tem um desses três nomes, caso tenha salta para “outro” e procura o próximo arquivo.

2- Abrir o arquivo

Usamos a syscall SYS_LE_ESCREVE para habilitar a leitura e a escrita do arquivo. Então abrimos o arquivo usando a syscall ABRIR e usamos a LSEEK para verificar o tamanho total do arquivo, em bytes. Assim, atribuímos ao ESI o tamanho do arquivo e fomos decrementando-o para percorrer todo o arquivo.

3- Verificar onde termina o cabeçalho

O código hexadecimal contém muitas informações que poderiam “confundir” o programa que resultaria em alguns erros de execução, ao tentar alterar a instrução desejada. Para evitar que ocorresse algum erro, desconsideramos o cabeçalho do arquivo executável.
Solução: criamos um loop chamado percorre, que procura uma seqüência de seis 0x90 que indica o inicio do cabeçalho e depois mais doze 0x90 que indicam o fim do cabeçalho. Só depois de encontrar o fim do cabeçalho é que começamos a pesquisar o código hexadecimal a ser alterado no arquivo.

4- Procurar as instruções desejadas

No loop “busca”, comparamos o byte lido com o valor 0x81, que corresponde a instrução ADD em assembly. Caso encontre essa instrução o programa salta para outro loop chamado segunda, que verifica se o próximo byte é 0xC1(ECX), 0xC2 (EDX) ou 0xC3 (EBX). Se não for nenhuma das três, salta para “busca” novamente. Caso contrário, salta para “encontrou”.

5- Alterar a instrução

Em “encontrou”, substituímos a instrução “Add Reg, Imediato” pelo código malicioso. O código é inserido através da syscall ESCREVER, que escreve o conteúdo da variável “vírus”. Portanto a partir da instrução ADD encontrada no executável o código malicioso é inserido. O código faz o programa entrar em um looping que escreve a mensagem “ HA” repetidamente e aloca espaço na pilha até ocasionar "estouro de pilha" gerando uma "falha de segmentação”.

6- Fechar o arquivo

Depois de alterar o código hexadecimal é necessário fechar o arquivo usando a syscall FECHAR. Exibimos também o nome do arquivo executável que foi alterado e a mensagem “Arquivo Infectado”. Para encerrar o programa, chamamos a syscall exit (definida por equ 1).

7- Considerações finais

Assim o vírus realiza as alterações solicitadas no arquivo executável. Quando supostamente o usuário vier a executar o arquivo infectado o comportamento do arquivo terá sido completamente modificado, conforme visto no item 5.

Um dos motivos por termos optado por inserir um JMP no final do código malicioso é que deveria haver um tratamento no executável, pois o código sobrepõe as instruções do arquivo, com isso se ao final do código inserido alguma instrução do arquivo original fosse quebrada ao meio haveria “falha de segmentação”. Como nossa última instrução é um JMP incondicional, o tratamento pode ser dispensado, pois o restante do arquivo nunca deverá ser executado.