**Attacking Applications Connecting to Services**

Applications that are connected to services often include connection strings that can be leaked if they are not protected sufficiently. In the following paragraphs, we will go through the process of enumerating and exploiting applications that are connected to other services in order to extend their functionality. This can help us collect information and move laterally or escalate our privileges during penetration testing.

**ELF Executable Examination**

The octopus\_checker binary is found on a remote machine during the testing. Running the application locally reveals that it connects to database instances in order to verify that they are available.

yovecio@htb[/htb]$ ./octopus\_checker

Program had started..

Attempting Connection

Connecting ...

The driver reported the following diagnostics whilst running SQLDriverConnect

01000:1:0:[unixODBC][Driver Manager]Can't open lib 'ODBC Driver 17 for SQL Server' : file not found

connected

The binary probably connects using a SQL connection string that contains credentials. Using tools like [PEDA](https://github.com/longld/peda) (Python Exploit Development Assistance for GDB) we can further examine the file. This is an extension of the standard GNU Debugger (GDB), which is used for debugging C and C++ programs. GDB is a command line tool that lets you step through the code, set breakpoints, and examine and change variables. Running the following command we can execute the binary through it.

yovecio@htb[/htb]$ gdb ./octopus\_checker

GNU gdb (Debian 9.2-1) 9.2

Copyright (C) 2020 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Type "show copying" and "show warranty" for details.

This GDB was configured as "x86\_64-linux-gnu".

Type "show configuration" for configuration details.

For bug reporting instructions, please see:

<http://www.gnu.org/software/gdb/bugs/>.

Find the GDB manual and other documentation resources online at:

<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".

Type "apropos word" to search for commands related to "word"...

Reading symbols from ./octopus\_checker...

(No debugging symbols found in ./octopus\_checker)

Once the binary is loaded, we set the disassembly-flavor to define the display style of the code, and we proceed with disassembling the main function of the program.

Code: assembly

gdb-peda$ set disassembly-flavor intel

gdb-peda$ disas main

Dump of assembler code for function main:

0x0000555555555456 <+0>: endbr64

0x000055555555545a <+4>: push rbp

0x000055555555545b <+5>: mov rbp,rsp

<SNIP>

0x0000555555555625 <+463>: call 0x5555555551a0 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>

0x000055555555562a <+468>: mov rdx,rax

0x000055555555562d <+471>: mov rax,QWORD PTR [rip+0x299c] # 0x555555557fd0

0x0000555555555634 <+478>: mov rsi,rax

0x0000555555555637 <+481>: mov rdi,rdx

0x000055555555563a <+484>: call 0x5555555551c0 <\_ZNSolsEPFRSoS\_E@plt>

0x000055555555563f <+489>: mov rbx,QWORD PTR [rbp-0x4a8]

0x0000555555555646 <+496>: lea rax,[rbp-0x4b7]

0x000055555555564d <+503>: mov rdi,rax

0x0000555555555650 <+506>: call 0x555555555220 <\_ZNSaIcEC1Ev@plt>

0x0000555555555655 <+511>: lea rdx,[rbp-0x4b7]

0x000055555555565c <+518>: lea rax,[rbp-0x4a0]

0x0000555555555663 <+525>: lea rsi,[rip+0xa34] # 0x55555555609e

0x000055555555566a <+532>: mov rdi,rax

0x000055555555566d <+535>: call 0x5555555551f0 <\_ZNSt7\_\_cxx1112basic\_stringIcSt11char\_traitsIcESaIcEEC1EPKcRKS3\_@plt>

0x0000555555555672 <+540>: lea rax,[rbp-0x4a0]

0x0000555555555679 <+547>: mov edx,0x2

0x000055555555567e <+552>: mov rsi,rbx

0x0000555555555681 <+555>: mov rdi,rax

0x0000555555555684 <+558>: call 0x555555555329 <\_Z13extract\_errorNSt7\_\_cxx1112basic\_stringIcSt11char\_traitsIcESaIcEEEPvs>

0x0000555555555689 <+563>: lea rax,[rbp-0x4a0]

0x0000555555555690 <+570>: mov rdi,rax

0x0000555555555693 <+573>: call 0x555555555160 <\_ZNSt7\_\_cxx1112basic\_stringIcSt11char\_traitsIcESaIcEED1Ev@plt>

0x0000555555555698 <+578>: lea rax,[rbp-0x4b7]

0x000055555555569f <+585>: mov rdi,rax

0x00005555555556a2 <+588>: call 0x5555555551d0 <\_ZNSaIcED1Ev@plt>

0x00005555555556a7 <+593>: cmp WORD PTR [rbp-0x4b2],0x0

<SNIP>

0x0000555555555761 <+779>: mov rbx,QWORD PTR [rbp-0x8]

0x0000555555555765 <+783>: leave

0x0000555555555766 <+784>: ret

End of assembler dump.

This reveals several call instructions that point to addresses containing strings. They appear to be sections of a SQL connection string, but the sections are not in order, and the endianness entails that the string text is reversed. Endianness defines the order that the bytes are read in different architectures. Further down the function, we see a call to SQLDriverConnect.

Code: assembly

0x00005555555555ff <+425>: mov esi,0x0

0x0000555555555604 <+430>: mov rdi,rax

0x0000555555555607 <+433>: call 0x5555555551b0 <SQLDriverConnect@plt>

0x000055555555560c <+438>: add rsp,0x10

0x0000555555555610 <+442>: mov WORD PTR [rbp-0x4b4],ax

Adding a breakpoint at this address and running the program once again, reveals a SQL connection string in the RDX register address, containing the credentials for a local database instance.

Code: assembly

gdb-peda$ b \*0x5555555551b0

Breakpoint 1 at 0x5555555551b0

gdb-peda$ run

Starting program: /htb/rollout/octopus\_checker

[Thread debugging using libthread\_db enabled]

Using host libthread\_db library "/lib/x86\_64-linux-gnu/libthread\_db.so.1".

Program had started..

Attempting Connection

[----------------------------------registers-----------------------------------]

RAX: 0x55555556c4f0 --> 0x4b5a ('ZK')

RBX: 0x0

RCX: 0xfffffffd

RDX: 0x7fffffffda70 ("DRIVER={ODBC Driver 17 for SQL Server};SERVER=localhost, 1401;UID=username;PWD=password;")

RSI: 0x0

RDI: 0x55555556c4f0 --> 0x4b5a ('ZK')

<SNIP>

Apart from trying to connect to the MS SQL service, penetration testers can also check if the password is reusable from users of the same network.

**DLL File Examination**

A DLL file is a Dynamically Linked Library and it contains code that is called from other programs while they are running. The MultimasterAPI.dll binary is found on a remote machine during the enumeration process. Examination of the file reveals that this is a .Net assembly.

C:\> Get-FileMetaData .\MultimasterAPI.dll

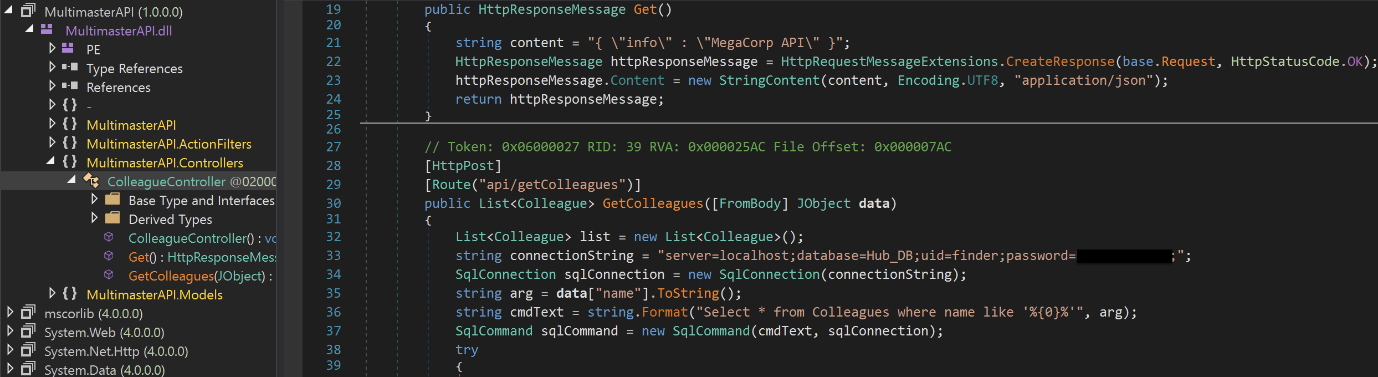
<SNIP>

M .NETFramework,Version=v4.6.1 TFrameworkDisplayName.NET Framework 4.6.1 api/getColleagues ! htt

p://localhost:8081\*POST Ò^ øJ ø, RSDSœ»¡ÍuqœK£"Y¿bˆ C:\Users\Hazard\Desktop\Stuff\Multimast

<SNIP>

Using the debugger and .NET assembly editor [dnSpy](https://github.com/0xd4d/dnSpy), we can view the source code directly. This tool allows reading, editing, and debugging the source code of a .NET assembly (C# and Visual Basic). Inspection of MultimasterAPI.Controllers -> ColleagueController reveals a database connection string containing the password.



Apart from trying to connect to the MS SQL service, attacks like password spraying can also be used to test the security of other services.