Part 1: Introduction to Exploit Development

This is the first part in a (modest) multi-part exploit development series. This part will just cover some basic things like what we need to do our work, basic ideas behind exploits and a couple of things to keep in mind if we want to get to and execute our shellcode. These tutorials will not cover finding bugs, instead each part will include a vulnerable program which needs a specific technique to be successfully exploited. In the fullness of time I intend to cover everything from 'Saved Return Pointer Overflows" to 'ROP (Return Oriented Programming)" of course these tutorials won't write themselves so it will take some time to get there. It is worth mentioning that these tutorials wont cover all the small details and eventualities; this is done by design to (1) save me some time and (2) allow the diligent reader to learn by participating.

I would like to give special thanks to Offensive Security and Corelan, thanks for giving me this amazing and painful addiction! !

(1) What we need

Immunity Debugger - Download
Immunity Debugger is similar to Ollydbg but it has python support which we will need to run plugin's to aid us with our exploit development. It's free; on the link just fill in some bogus info and hit download.

Mona.py - Download
Mona is an amazing tool with tons of features which will help us to do rapid and reliable exploit development. I won't be discussing all the options here, we'll get to them during the following parts of the tutorial. Download it and put it in Immunity's PyCommands folder.

Pvefindaddr.py - Download

Pvefindaddr is Mona's predecessor. I know it's a bit outdated but it's still useful since there are some features that haven't been ported to Mona yet. Download it and put it in Immunity's PyCommands folder.

Metasploit Framework - Download

We are going to use the Metasploit Framework extensively. Most of all we are going to be generating shellcode for our exploits but we are also going to need a platform that can receive any connections we might get back from the programs we are exploiting. I suggest you use **Backtrack** since it has everything we need but feel free to set up metasploit in any way you see fit.

Virtualization Software

Basically there are two options here VirtualBox which is free and Vmware which isn't. If its possible I would suggest using Vmware; a clever person might not need to pay for it ;)). Coupled with this we will need several (32-bit) operating systems to develop our exploits on (you will get the most use out of WindowsXP PRO SP3 and any Windows7).

(2) Overflows

For the purpose of these tutorials I think it's important to keep things as simple or difficult as they need to be. In general when we write an exploit we need to find an overflow in a program. Commonly these bugs will be either Buffer Overflows (a memory location receives more data than it was meant to) or Stack Overflows (usually a Buffer Overflow that writes beyond the end of the stack). When such an overflow occurs there are two things we are looking for; (1) our buffer needs to overwrite EIP (Current Instruction Pointer) and (2) one of the CPU registers needs to contain our buffer. You can see a list of x86 CPU registers below with their separate functions. All we need to remember is that any of these registers can store our buffer (and shellcode).

```
EAX - Main register used in arithmetic calculations. Also known as accumulator, as it holds results
      of arithmetic operations and function return values.
EBX - The Base Register. Pointer to data in the DS segment.  Used to store the base address of the
      program.
ECX - The Counter register is often used to hold a value representing the number of times a process
      is to be repeated. Used for loop and string operations.
EDX - A general purpose registers. Also used for I/O operations. Helps extend EAX to 64-bits.
ESI - Source Index register. Pointer to data in the segment pointed to by the DS register.  Used as
      an offset address in string and array operations. It holds the address from where to read data.
EDI - Destination Index register. Pointer to data (or destination) in the segment pointed to by the
      ES register.  Used as an offset address in string and array operations. It holds the implied
      write address of all string operations.
EBP - Base Pointer. Pointer to data on the stack (in the SS segment).  It points to the bottom of the
      current stack frame. It is used to reference local variables.
```

```
ESP - Stack Pointer (in the SS segment). It points to the top of the current stack frame. It is used
      to reference local variables.
EIP - Instruction Pointer (holds the address of the next instruction to be executed)
```

(3) How does it work?

Basically (1) we get a program to store an overly long string, (2) this string overwrites EIP and part of it is stored in a CPU register, (3) we find a pointer that points to the register that contains our buffer, (4) we put that pointer in the correct place in our buffer so it overwrites EIP, (5) when the program reaches our pointer it executes the instruction and jumps to the register that contains our buffer and finally (6) we place our shellcode in the part of the buffer that is stored in the CPU register. In essence we hijack the execution flow and point it to an area of memory that we control. If we are able to do that we can have to remote machine execute any instructions we place there. This is a bit simplistic but it should give you a basic idea of how exploits work.

Comments

There are no comments posted yet. Be the first one!

Post a new comment

Enter text right here!

Name

Email

Displayed next to your comments.

Not displayed publicly.

Subscribe to None

Submit Comment