Week 14. Identifying Executables, Functions, Outside Communication

---

Identifying Executables - Ch 5

- Modern code is NOT minimalistic.
  - Not because the programmers are writing a lot, but because a lot of libraries are commonly linked statically to executable files. If all external libraries were shifted into an external DLL files, the world would be different. (Another reason for C++ are the STL and other template libraries.)
  - Thus, it is very important to determine the origin of a function, if it is from standard library or well-known library (like Boost1, libpng2), or if it is related to what we are trying to find in the code.
  - It is just absurd to rewrite all code in C/C++ to find what we're looking for.
    One of the primary tasks of a reverse engineer is to find quickly the code he/she needs.
    - If you can identify that they use a library that is already compromised / has known exploits for, you might be able to use a function from that as a point of entry.
      - As in cybersecurity, hackers (reverse engineers included) just want one way into the system, and then they can pivot.
  - **NEVER** be afraid to google code that looks familiar
    - Constants, Text Strings, etc.
    - Google is every good reverse engineers friend, as it is for software developers.
- Here are a few tips for identifying what code your executable is written in and what compiled it:
  - **MSVC (microsoft visual c++)**

Marketing ver.	Internal ver.	CL.EXE ver.	DLLs imported	Release date
6	6.0	12.00	msvcrt.dll msvcp60.dll	June 1998
.NET (2002)	7.0	13.00	msvcr70.dll msvcp70.dll	February 13, 2002
.NET 2003	7.1	13.10	msvcr71.dll msvcp71.dll	April 24, 2003
2005	8.0	14.00	msvcr80.dll msvcp80.dll	November 7, 2005
2008	9.0	15.00	msvcr90.dll msvcp90.dll	November 19, 2007
2010	10.0	16.00	msvcr100.dll msvcp100.dll	April 12, 2010
2012	11.0	17.00	msvcr110.dll msvcp110.dll	September 12, 2012
2013	12.0	18.00	msvcr120.dll msvcp120.dll	October 17, 2013

  - If the executable imports msvcp*.dll it is probably a C++ program.
- **GCC**
  - If compiled via windows:
    - Cygwin
      - cygwin1.dll is often imported.
    - MinGW
      - msvcrt.dll may be imported.
- Name Mangling - pg.542
  - A C++ class may contain several methods sharing the same name but having different arguments—that is polymorphism. And of course, different classes may have their own methods with the same name.
  - Name mangling enable us to encode the class name + method name + all method argument types in one ASCII string, which is then used as an internal function name. That's all because neither the linker, nor the DLL OS loader (mangled names may be among the DLL exports as well) knows anything about C++ or OOP23.
  - MSVC
    - ? Prefix

    - GCC
        - _Z symbols
    - **Fortran**
        - libifcoremd.dll, libifportmd.dll and libiomp5md.dll (OpenMP support) may be imported. libifcoremd.dll has a lot of functions prefixed with for_, which means Fortran.
    - Others are listed in the book
- How is this useful?
    - Knowing what language and or what compiler compiled the code gives you an advantage in understanding the executable. As we have seen, different compilers and even different compiler versions compile code into functionally similar, but visually distinct chunks of machine code.
        - This helps us get into the mindset of looking for patterns that are common or typical with that compiler.

Functions

- It's often advisable to track function arguments and return values in debugger or DBI.
- Watching inputs and outputs of unknown functions will help you understand what it does.
- Be keenly aware of JMP operators! These are the branches in the path for the code, and following them is the key to understanding.

Communicating outside the program

- Win32
    - Files and registry access: for the very basic analysis, **Process Monitor** utility from SysInternals can help.
    - For the basic analysis of network accesses, **Wireshark** can be useful.
        - But then you will have to look inside anyway.

- The first thing to look for is which functions from the OS's APIs and standard libraries are used. If the program is divided into a main executable file and a group of DLL files, sometimes the names of the functions in these DLLs can help.

- *NIX
  - **Strace/Trace** is an extremely useful tool
    - It is used to monitor and tamper with interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state. The operation of strace is made possible by the kernel feature known as ptrace.
    - `man (s)trace`
    - `strace -p [processnumber]`
  - For the basic analysis of network accesses, **Wireshark** can be useful.
    - But then you will have to look inside anyway.
  - The first thing to look for is which functions from the OS's APIs and standard libraries are used.
    - If the program is divided into a main executable file and a group of DLL files, sometimes the names of the functions in these DLLs can help.