IT Security Principles

IT 444 – Network Security

Computer Memory

- An electronic mechanism to store and retrieve data
- The smallest amount of data is 1 bit (1 or 0 in memory)
  - 4 bit = 1 nibble
  - 8 bits = 2 nibbles = 1 byte
  - 2 bytes = a word
  - 2 words = double word DWORD

Computer Memory

- Many types of memory, but we are focusing on random access (RAM) and registers
  - Registers: special forms of memory embedded within processor
  - RAM: volatile. Data is lost from RAM when computer is off
- Modern Intel & AMD has the memory of 32 bit or 48 bit addressable (32 or 48 bit wide)

Memory segmentation

- Each process needs to have access to its own area in memory.
  - Memory is broken into segments
  - Registers store and keep track of the current segment a process maintains
  - Offset registers keep track of where critical piece of data is kept in the segment
- Each running process gets its own virtual address space...

Memory segmentation

- A basic 32-bit Windows process gets 4GB - 2GB is assigned to the user-mode side and 2GB is assigned to the kernel-mode side of the process
- A small portion of this virtual space within each process is mapped to physical memory

Program in Memory

- When processes are loaded into memory, they are basically broken into many small sections
  - The .*text* section, contains the machine instructions to get the task done
  - The .*data* section is used to store global initialized variables (int a =0;)
  - The *below stack section (.bss)* is used to store certain types of global uninitialized variables (int a;)

Program in Memory

- The *heap* section is used to store dynamically allocated variables and grows from the lower-addressed memory to the higher-addressed memory
- The *stack* section is used to keep track of function calls (recursively) and grows from the higher-addressed memory to the lower-addressed memory on most systems

Memory space of a process

- Buffer: is a storage place used to receive and hold data until it can be handled by a process
- Done by allocating the memory within the .data or .bss section of the process's memory
- The buffer may hold any predefined type of data

String and Pointer

- *strings* are just continuous arrays of character data in memory
- The string is referenced by the address of the 1st char
- The string is ended by a null character (**`\0`** in C)
- Pointers are special pieces of memory that hold the address of other pieces of memory
- Keeping track of the items' locations in memory is by changing the info in pointers

What is Fuzzing

- One of the fastest ways to get into vulnerability research is through software testing.
- Fuzzing is a class of software and hardware testing in which the data used to perform the testing is randomly generated
- Doesn’t require any knowledge about the internal workings of software or the structure of the input data

What is Fuzzing

- Types of Fuzzers
  - Mutation fuzzers
  - Generation fuzzers
  - Genetic or evolutionary fuzzers

Fuzzers

- **Mutation fuzzers:** changing the input data in a random way
  - The mutated data is then used as input for the target software in order to try and trigger a software crash
- **Generation fuzzers:** white box fuzz testing –prior knowledge of the internal workings of the protocol
  - Able to generate test cases based on data models that describe the structure of the data or protocol

Fuzzers

- the main problems with generation fuzzers is writing data models
- the availability of specifications and documentation still requires significant effort to correctly translate to a fuzzing model

Fuzzers

- **Genetic fuzzing:** also called evolutionary fuzzing
  - The tool determines the best set of input tests, based on maximizing code coverage over time
  - The fuzzer makes notice of input mutations that reach new code blocks and saves those mutated inputs to the body (corpus) of tests
  - The fuzzing tool can learn in a “survival of the fittest” manner—thus the term *genetic* or *evolutionary fuzzing*

Crash Analysis

- There should be some logs for the target application crashes
- Sample file or data records that can be used to reproduce the crash
- Application crash log files can be collected in many ways
  - debugger will collect information about the CPU context, which will be stored along with the crash sample file
- Crash logs provide a great first step in filtering and grouping crashes into unique vulnerabilities

Crash Analysis

- When an application crash is detected, many custom scripts can be run that collect specific types of information.
- The easiest way to implement such scripts is by extending the debugger
- Using Peach as the framework produces some nice benefits when you're dealing with crashes. Peach uses ***WinDbg*** and the **!exploitable** extension to gather contextual information about a crash and to be able to perform some crash clustering

Binary diffing

- When changes are made to compiled code such as libraries, applications, and drivers, the delta between the patched and unpatched versions can offer an opportunity to discover vulnerabilities
- The most common target of binary diffs are Microsoft patches
- Various tools are available to simplify the process of binary diffing

Diffing

- ***Application:*** new versions – new features, code changes, new security control or vulnerability fixes
  - Identifying code changes related to vulnerability fixes is dependent on limited disclosures
  - Many organizations choose to release minimal information as to the nature of a security patch
  - The more clues we can obtain from this information, the more likely we are to discover the vulnerability
- ***Patch diffing:*** organizations don't patch their systems quickly. Attackers and penetration testers can compromise these systems with publicly disclosed or privately developed exploits

Binary Diffing Tools

- Manually analyzing is slow, not effective
- Free tools are available:

  `zynamics bindiff`

  `turbodiff`

  `patchdiff2`

  `darungrim`

  `diaphora`

- You may experience *different* results when using each tool against the *same* input files

Microsoft Patch

- Patches are acquired from the ***Microsoft Security Tech-Center*** site
- Patches are obtained by ***WSUS*** or ***Control Panel***
- Some updates are the result of a publicly discovered vulnerability...
- ...whereas the majority are through some form of coordinated private disclosure

Microsoft Patch

- Find the update info for the *last* time the files were patched
- This is important to note because you always want to diff the versions *closest together* so that any changes to functions are associated with the CVEs in which you are interested
- We can use tools such as “PatchExtract” for easy extraction
- PatchExtract is a *PowerShell* script that uses the Microsoft “expand” tool to extract many files contained within the downloaded MSU files

Diffing a MS Patch

- When extracting the patch, we can determine the xyz.dll was updated.
- The first step is to disassemble these using the tool IDA (or alike) and perform a diff.
- From the result, we can find:
  - The changed file
  - What DLL is being loaded . The goal is to identify what application desire this DLL

Diffing a MS Patch

- Use the “Process Monitor” tool to find the path to the DLL file found previously
- Try Microsoft applications and use “Process Monitor” to verify that the dll is loaded
- If we can craft a malicious DLL with msfvenom as our payload, we should be able to get a remote session with the vulnerable Windows system.