Topics of Ch-4

- System Monitoring Tools
- Dynamic Tracing: System Calls, Filesystem, and Registry
- Compiler Issues
- Debuggers
  - OllyDbg
  - WinDbg
- Disassemblers
  - IDA Pro
  - Decompilers
- Memory Analysis to Support Reverse Engineering
  - DRAM Acquisition
  - Extraction of Malware

Dynamic Tracing

- **DTrace** is a comprehensive dynamic tracing framework originally created by Sun Microsystems for troubleshooting kernel and application problems on production systems in real time. Originally developed for Solaris, it has since been released under the free Common Development and Distribution License (CDDL)
- DTrace can be used to get a global overview of a running system, such as the amount of memory, CPU time, filesystem and network resources used by the active processes. It can also provide much more fine-grained information, such as a log of the arguments with which a specific function is being called, or a list of the processes accessing a specific file.
- As a kind of post-modern advanced debugging technology, dynamic tracing allows software engineers to answer some tricky problems about software systems, such as high CPU or memory usage, high disk usage, long latency, or program crashes. All this can be detected at a low cost within a short period of time, to quickly identify and rectify the problems.

Dynamic Tracing

- Dynamic tracing usually works based on the operating system kernel level, where the “supreme being of software” has complete control over the entire software world. With absolute authority, the kernel can ensure the “queries” targeted at the software system will not influence the latter’s normal running.
- That is to say, those queries must be secure enough for wide use on production systems. Then, there arises another question concerning how a query is made if the software system is regarded as a special “database”. Clearly, the answer is not SQL.

System Calls, Filesystem, and Registry

- Monitoring system calls (syscalls) and analyzing system behavior can help you debug your products and improve their performance, security, and compliance. However, monitoring syscalls in Windows has its challenges due to a lack of built-in tools and the need for niche knowledge of reverse engineering and application behavior analysis.
- A syscall is a mechanism enabling programs to request various types of services and tasks from the operating system's kernel. System calls provide a secure and controlled way for user-level programs to access system resources without compromising the stability or security of the operating system (OS). When executing a requested operation, a system call transfers control between user and kernel modes.
- In an operating system, there are two code environments:
  - Kernel code (ring 0) that runs with full privileges
  - User code (ring 3) that runs with limited privileges
- Between these environments, there's a system call interface.

System Calls, Filesystem, and Registry

- he divide between ring 0 and ring 3 — the kernel and user levels — is the best point at which to observe an application's behavior and raw actions. When code has reached the level of system calls, the application can't obfuscate its actions. For example, if user-level code calls the **CreateFile()** function covertly, you'll still be able to detect this by monitoring system calls, as you'll see the execution of the **NtCreateFile** syscall.
- For the majority of popular architectures, including x86, AMD64, ARM64, and PowerPC, the approach to working with system calls is the same:
  - **At the user level**, a set of system APIs act as wrappers over system calls.
  - **At the kernel level**, a kernel API implements handlers for system calls and is used by kernel drivers.
- In some architectures, system calls are called *system services*.

System Calls, Filesystem, and Registry

- **Common examples of syscalls in Windows include:**
  - File input/output (I/O) operations, such as reading from or writing to a file
  - Process management, such as creating a new process or terminating an existing one
  - Memory management, such as allocating memory
  - Interprocess communication, such as sending or receiving messages between processes
  - Device driver operations, such as sending commands to a hardware device
- **Common reasons for monitoring syscalls include:**
  - Debugging — Track down issues in your application, identify their root cause, and fix bugs quickly.
  - Performance optimization — Identify bottlenecks and optimize problematic sections of code to improve overall performance.
  - Security — Detect suspicious, potentially malicious behavior and take steps to prevent it.
  - Compliance — Ensure that an application complies with relevant requirements by analyzing the way the application accesses and uses specific types of data.

System Calls, Filesystem, and Registry

- **Process and Performance Analysis**
- Beyond system-level monitoring, DTrace is particularly adept at dissecting individual processes. It can provide detailed information about process execution, including CPU and memory usage, helping to pinpoint performance bottlenecks or memory leaks. This granular level of detail is invaluable for performance tuning and debugging complex software issues.
- **Customizability and Flexibility**
- One of the most powerful aspects of DTrace is its customizability. With a scripting language based on C syntax, DTrace allows the creation of customized scripts to probe specific aspects of system behavior. This flexibility means that it can be adapted to a wide range of debugging scenarios, making it a versatile tool in a developer's arsenal.
- **Real-World Applications**
- In practical terms, DTrace can be used to diagnose elusive performance issues, track down resource leaks, or understand complex interactions between different system components. For example, it can be used to determine the cause of a slow file operation, analyze the reasons behind a process crash, or understand the system impact of a new software deployment.

Compiler Issues

- **Compilation Process**
- The C compilation process is a series of steps that **converts human-readable C code into machine-executable** binary code.
- The general steps involved in the C compilation process are:
  - Preprocessing
  - Compilation
  - Assembly
  - Linking
  - Loading
  - Execution
- most compilers will give three types of compile-time alerts: compiler warnings, compiler errors, and linker errors.

Compiler Issues

- Although you don't want to ignore them, compiler warnings aren't something severe enough to actually keep your program from compiling. Usually, compiler warnings are an indication that something might go wrong at runtime.
- Compiler Version Mismatch Errors
- Error is present in custom code specified as an S-Function block or in **Custom Code**. For example, the code refers to a header file that the compiler cannot find.
- The model includes a block, such as a device driver block, which is not intended for use with the currently selected system target file.
- A linker error about an undefined reference to the data appears when the model build generates an executable program from the model reference hierarchy and these conditions are true:
  - You represent signal, state, or parameter data by creating a data object such as ”Simulink.Signal” You use the object in a model reference hierarchy.
  - You use a custom storage class with the data object. Custom storage classes require Embedded Coder.
  - You set the owner of the object to a model that does not directly access the data

DRAM Acquisition

- DRAM stands for “dynamic random access memory,” and it’s a specific type of RAM (random access memory). All computers have RAM, and DRAM is one kind of RAM we see in modern desktops and laptops. DRAM was invented in 1968 by Robert Dennard and put to market by Intel® in the ‘70s.
- DRAM (dynamic random access memory) is a type of semiconductor memory that is typically used for the data or program code needed by a computer processor to function.
- DRAM is a common type of random access memory (RAM) that is used in PCs, workstations and servers. Random access allows the PC processor to access any part of the memory directly rather than having to proceed sequentially from a starting place.
- RAM is located close to a computer's processor and enables faster access to data than storage media such as hard disk drives and solid-state drives.
- As you use your computer, it needs to recall data and programming code for the CPU to process. RAM provides a way for the computer to use, rewrite, and temporarily save this data and code in real-time. Because the transistors need electricity to work, however, anything stored here disappears when you turn your PC off. That's why it's considered “volatile.”

DRAM Acquisition

- DRAM, or Dynamic Random Access Memory, is a **temporary memory bank for your computer** where data is stored for quick, short-term access. When you perform any task on your PC, such as launching an application, the CPU on your motherboard pulls program data from your storage device (SSD/ HDD) and loads it onto the DRAM.
- Since **DRAM is significantly faster** than your storage devices (even SSDs), the CPU can read this data quicker, resulting in better performance. The speed and capacity of your DRAM help determine how fast applications can run and how efficiently your PC can multitask. Hence, having faster and higher capacity DRAM is always beneficial.
- DRAM is the **most common type of RAM** we use today. The RAM DIMMs (dual in-line memory modules) or sticks that we install in our computers are, in fact, DRAM sticks. But what exactly makes DRAM dynamic? Let’s find out!

DRAM vs SRAM

- There are two main classifications of primary memory — **DRAM** (Dynamic Random Access Memory) and **SRAM** (Static Random Access Memory). While we have learned what DRAM is and how it works, how does it compare to SRAM?
- SRAM uses a six-transistor memory cell to store data, as opposed to the transistor and capacitor pair approach taken by DRAM. SRAM is an on-chip memory typically **used as cache memory for CPUs**.
- It's considerably faster and more power efficient than most other types of RAM, including DRAM. However, it is also significantly more expensive to produce and isn't user-replaceable/ upgradeable. DRAM, on the other hand, is often user replaceable

DRAM vs SRAM

DRAM	SRAM
It uses capacitors to store data	It uses transistors to store data
Capacitors need constant refreshing to retain data	Doesn’t need refreshing as it doesn’t use capacitors to store data
Has slower speeds than SRAM	Significantly faster than DRAM
Cheaper to manufacture	Very expensive
DRAM devices are high-density	SRAM is low-density
Used as main memory	Used as cache memory for CPUs
Relatively lower heat output and power consumption than SRAM	High heat output and power consumption

DRAM Acquisition

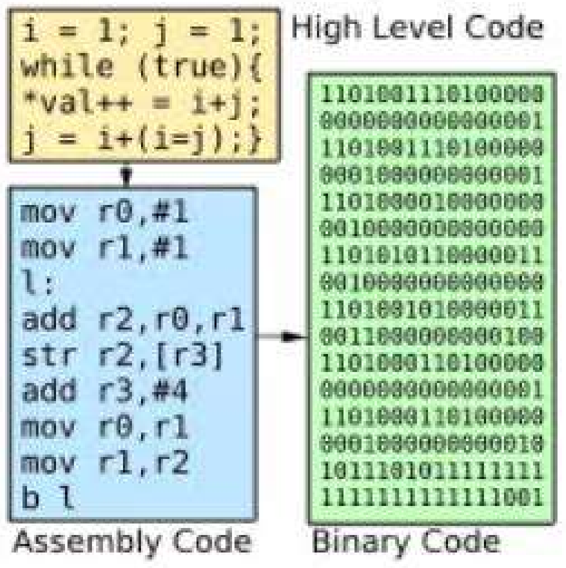
- Types of DRAM
  - SDRAM
  - DDR SDRAM
  - ECC DRAM
  - DDR2, DDR3, AND DDR4
- **Types of DRAM packages**
- There are two main types of DRAM packaging: single inline memory module (SIMM) and dual inline memory module (DIMM).
- Single inline memory module packaging is considered obsolete now and was used in the 1980s to 1990s. SIMMs came in 30 and 72 pin sets and typically had 32-bit data transfer rates. DIMMs, on the other hand, are commonly used now and have pins on both sides of the chip.
- DIMMs commonly have 168 pin connectors -- or more -- and support a 64-bit data transfer rate.

DRAM Acquisition

- **Advantages**
  - The main advantages of DRAM include the following:
  - Its design is simple, only requiring one transistor.
  - The cost is low in comparison to alternative types of memory, such as SRAM.
  - It provides higher density levels.
  - More data can be stored using DRAM.
  - Memory can be refreshed and deleted while a program is running.
- **Disadvantages**
  - The main disadvantages of DRAM include the following:
  - Memory is volatile.
  - Power consumption is high relative to other options.
  - Manufacturing is complex.
  - Data in storage cells needs to be refreshed.
  - It is slower than SRAM.

Decompilers, Debugger & Disassemblers

- **Decompilers** reverse binaries into higher-level languages, like C++
- **Disassemblers** reverse binaries into assembler language
- **Debuggers** allow you to view and change the state of a running program
- **Hex Editors** allow you to view and edit the contents of a binary



```
i = 1; j = 1;
while (true){
*val++ = i+j;
j = i+(i=j);}
```

```
mov r0,#1
mov r1,#1
l:
add r2,r0,r1
str r2,[r3]
add r3,#4
mov r0,r1
mov r1,r2
b l
```

```
1101001110100000
0000000000000001
1101001110100000
0001000000000001
1101000010000000
0010000000000000
1101010110000011
0010000000000000
1101001010000011
0011000000000100
1101000110100000
0000000000000001
1101000110100000
0001000000000010
1011101011111111
1111111111111001
```

Decompilers, Debugger & Disassemblers

- **Disassemblers**
- Convert the application's machine code to assembly code. Used in static analysis. The static analysis is a code interpretation technique that enables understanding of the program's behavior without running it. IDA Pro is an example of a disassembler.
- **Decompilers**
- Utilized to convert binary code to high-level code (pseudocode). Generate high-level code that is shorter and simpler to read and comprehend.

Decompilers, Debugger & Disassemblers

- **Debuggers**
- In addition to code disassembly, debuggers enable the reverser to execute the target program in a controlled manner, i.e., rather than executing the entire binary, the reverser can execute a specific instruction or function. While a program is running, you can view and modify its execution flow to gain insight into its functionality.
- Permits the reverser to conduct a dynamic analysis by controlling specific aspects of the program, such as memory areas, during its execution. This assists in comprehending the program’s functionality and its effects on a system or network.
- Examples of debuggers are Ollydbg, Immunity Debugger, x64dbg, GDB and Windbg.

Extraction of Malware

- **Malware Infection Vectors**
- Phishing Emails
- Infected Websites
- Pirated Software
- Contaminated Removable Drives
- Software Vulnerabilities
- Network Intrusions
- Social Engineering Manipulation

Signs of Malware Infection on Windows

- Degradation in system performance
- Slow boot times
- Programs freezing or crashing frequently
- Unknown processes running in Task Manager
- Mysterious network activity
- Disabled security software and antivirus tools
- Modified, missing, or corrupted files
- Changes to the Windows registry
- New administrator accounts
- Shortcut files appearing on the desktop
- Popups for fake software upgrades or scans

How to Safely Remove Malware on Windows

STEP 1: Use Rkill to terminate malicious processes
STEP 2: Uninstall malicious programs from Windows
STEP 3: Reset browsers back to default settings
STEP 4: Use Malwarebytes to remove for Trojans and Unwanted Programs
STEP 5: Use HitmanPro to remove Rootkits and other Malware
STEP 6: Use AdwCleaner to remove Malicious Browser Policies and Adware
STEP 7: Perform a final check with ESET Online Scanner
STEP 8: Run the System File Checker (SFC) tool
STEP 9: Run the Disk Check tool

Practical: https://malwaretips.com/blogs/malware-removal-guide-for-windows/