Reverse Engineering
Class 4

Malware Analysis

martin.uy
Open by default.

Malware Analysis

WARNING

Executing real malware during dynamic analysis has risks. An isolated environment is recommended.

If an hypervisor is used, use the most updated version and add as few peripherals as possible (I.e. avoid 3D acceleration).

Malware Analysis

- Some generic techniques used by binary malware will be seen in this class, from analysis and development perspectives
  - Each case is different (purpose and target)
  - Not every malware is binary malware
  - This area is constantly evolving: new APIs, new functionality, new anti-virus (AV) and detection heuristics

Malware Analysis

- Components
  - Agent
    - Executes operations in the attacked target (endpoint): computing, data stealing, data hijacking, etc.
    - Different autonomy levels are possible
      - Respond to commands interactively executed (lightweight and generic agent)
      - Operate autonomous using sophisticated data ex-filtration techniques (intelligent and specific agents)
        - Avoid “noise” in both the endpoint and network

Malware Analysis

- Components
  - Agent
    - It desirable to be of small size and Position-Independent-Code, to execute even when injected into a process (so process memory can be directly read)
    - Exposed to anti-virus and to bring traces of the attacker
    - If binary, designed to target a specific platform (operating system and architecture)

Malware Analysis

- Components
  - Command and Control center (C&C)
    - Consolidates commands and information from different agents
    - No exposure to anti-virus
  - Communication channel
    - Encrypted communications
    - Covert-channels
  - Agents and their C&C are usually called “botnet”

Malware Analysis

- Defense
  - Prevention is complex: organizations need to work on detection
  - Defense solutions were traditionally focused on fingerprint analysis (on endpoints or network). Today that's not enough
  - Dynamic analysis or "user behavior" is necessary

Malware Analysis

- What are the pros and cons of a TCP/UDP/IP communications channel from the C&C to the agent?

Malware Analysis

- TCP/UDP/IP to agent
  - What's the agent IP address? Does it change? (dynamic IP) Is it behind a NAT? is it an internal IP? Is there port-forwarding?
  - A firewall can easily block a TCP connection or an incoming UDP packet, unless it's sent to a known port of a known host
  - A process has to be listening on a port (high, if agent couldn't elevate privileges) and that's suspicious

Malware Analysis

- TCP/UDP/IP to agent
  - Agent has to encrypt ex-filtrated data (avoid suspicion to an IDS)
  - Provides good bandwidth to ex-filtrate huge amounts of data
  - C&C has control to initialize and finalize the channel. Agent does not need to be continuously polling for reconnection

Malware Analysis

- TCP/UDP/IP to agent
  - Port knocking. If agent achieved privileges to read raw data from the network interface, it can sense for a key (example: sequence of TPC SYNs to different ports). If key happens, it hast to listen to the C&C on a certain port
    - Avoid “always listening” to be stealthier
  - C&C can change its IP address (to protect the attacker)

Malware Analysis

- What are the pros and cons of a TCP/UDP/IP channel from the agent to the C&C?

Malware Analysis

- TCP/UDP/IP to the C&C
  - What's the C&C IP address? How does the agent know it? Is a DNS query needed?
    - A fixed C&C IP address can easily make an agent useless
    - A DNS query can be suspicious (domains blacklisting)
    - Attacker can loose control over the domain and make the botnet useless
    - There is not enough flexibility for the C&C

Malware Analysis

- TCP/UDP/IP to the C&C
  - Advanced agents can search for headlines in news and try to connect to a domain combining those words
  - Attacker registers the domain the same day and connectivity is reestablished

Malware Analysis

- TCP/UDP/IP to the C&C
  - There are no issues with NAT if attacked endpoint has Internet connectivity
  - C&C can used a known port, generally allowed for outgoing connections (I.e. 80, 443)
    - However, a firewall with deep inspection can find an unexpected protocol on a known port and thus consider the traffic suspicious

Malware Analysis

- TCP/UDP/IP to the C&C
  - Agent does not need to be always listening, but decides when to establish the connection. Continuously polling may be suspicious
  - Agent has to encrypt ex-filtrated data
  - Channel with good bandwidth

Malware Analysis

- What are the pros and cons of an HTTP/S, DNS, ICMP or other covert-channels the agent to the C&C?

Malware Analysis

- HTTP/S, DNS, ICMP and other covert-channels
  - Protocol abuse. I.e. malformed DNS queries
  - Usually depend on an attacker domain resolution (o are attached to a fixed IP address)
  - Has all the advantages of being initialized as outgoing traffic by the agent
  - Usually allowed in firewalls and less suspicious if implemented emulating human behavior (I.e. be careful with transfers cadence)
  - Less bandwidth

Malware Analysis

- Multi-staging / bootstrapping
  - Malware is frequently deployed in multiple stages
    - Script that downloads a binary
    - Exploiting payload (small size) that ends up downloading a complete payload
    - First stage can do some system profiling. Example: previous infection, operating system, architecture, etc. and take decisions regarding how to get and load the next stage

Malware Analysis

- Multi-staging / bootstrapping
  - May be interesting to avoid persistent traces: only the bootstrapper is persisted and not the agent (business?) logic
  - Do not write the agent to the file system to evade anti-virus
    - Anti-virus install kernel hooks to detect file system operations and act upon

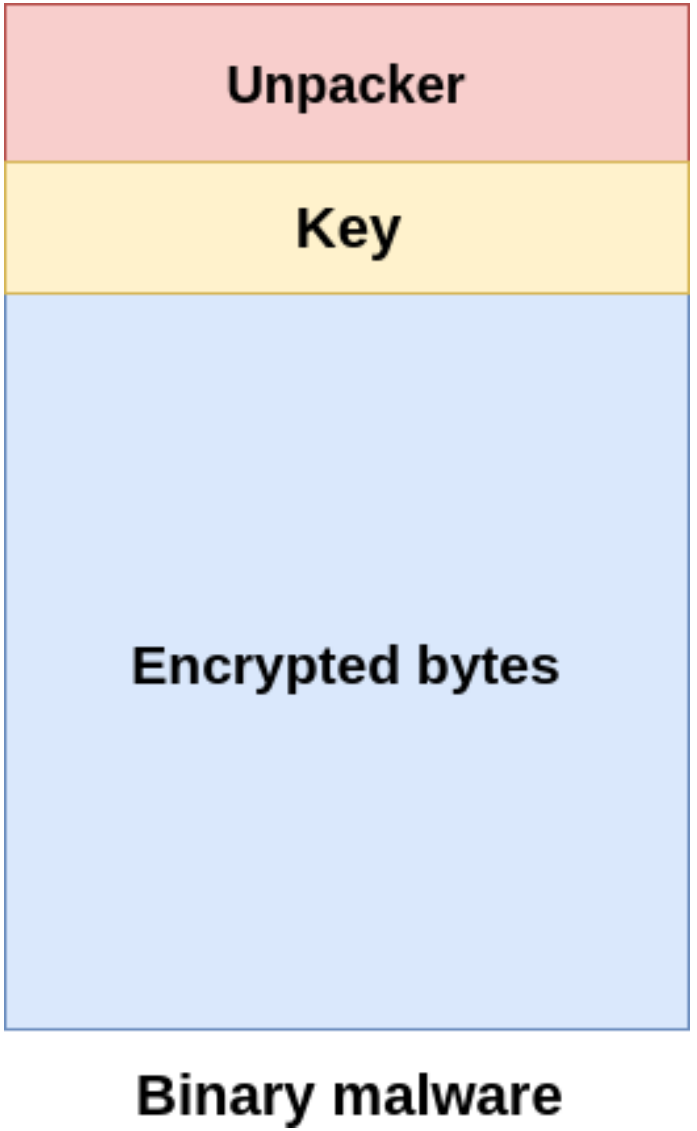
Malware Analysis

- The simplest strategy to start understanding a malware is analyzing strings inside
  - I.e. URL to the 2nd stage
  - `strings bin` (Linux)
  - Advanced malware does not expose strings (data, imported functions symbols, etc.) nor code in plain: they are encrypted or obfuscated to avoid most basic static analysis techniques

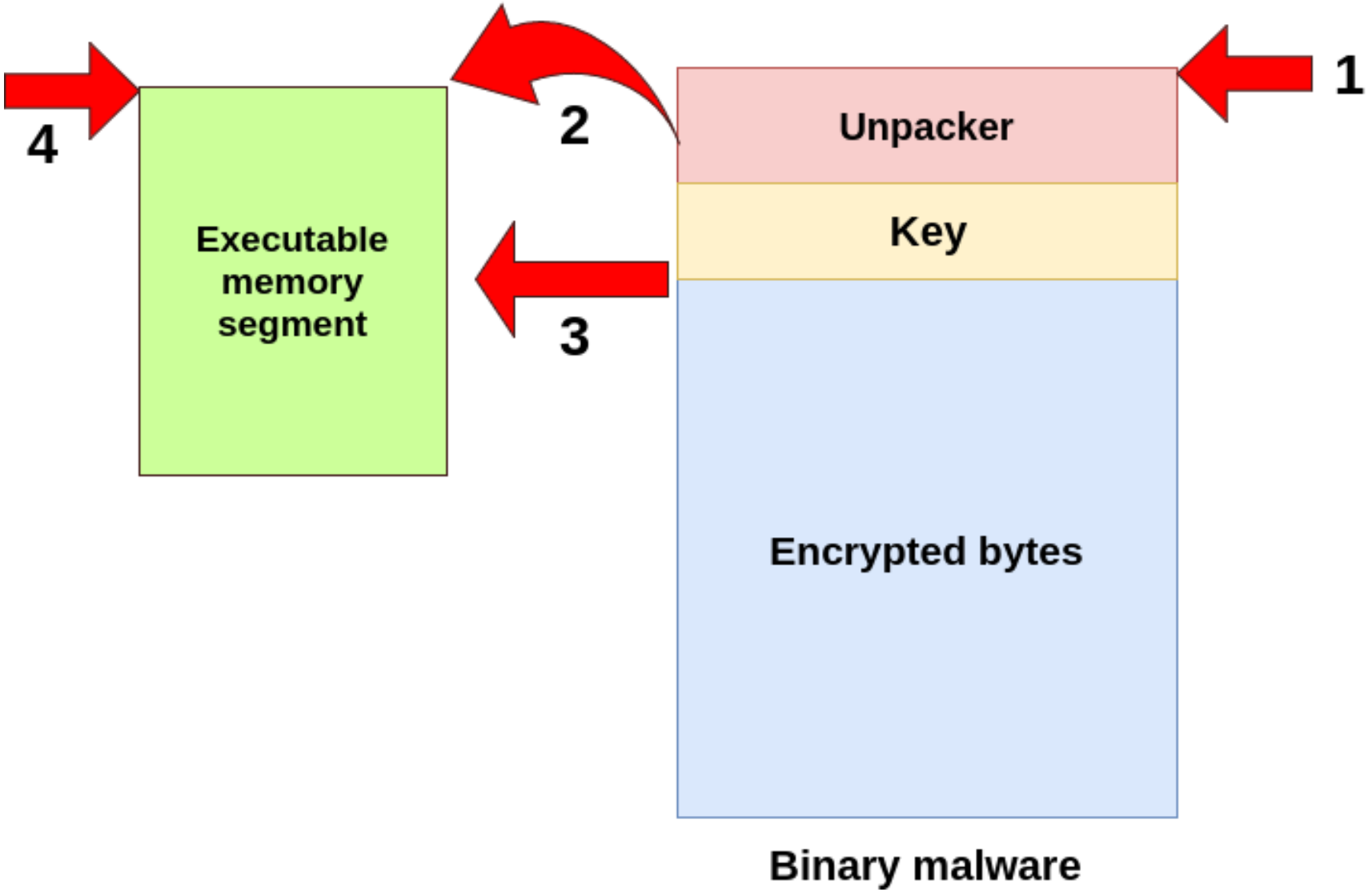
Malware Analysis

- Unpacking
  - When execution begins, malware has to unpack itself (bootstrapping)
    - Totally or partially
  - Execution starts at the unpacker, which uses a key inside the binary or obtained from a different source
    - Trivial cryptography: only for obfuscation and basic anti-virus/IDS fingerprinting evasion

Malware Analysis



Malware Analysis



Malware Analysis

1. Unpacker begins execution (binary entry-point)
2. Unpacker allocates an executable memory segment (write and, afterwards, execution permissions)
3. Unpacker reads or gets the key and encrypted bytes. After decryption, writes plain bytes in previously allocated memory segment
4. Unpacker jumps to execute decrypted bytes

Malware Analysis

- Isn’t suspicious that a process allocates memory, writes some bytes, changes permissions to execution and jumps there? What would make a legitimate use-case for that?

Malware Analysis

- Isn’t suspicious that a process allocates memory, writes some bytes, changes permissions to execution and jumps there? What would make a legitimate use-case for that?

JIT compilers (OpenJDK, Flash, etc.) 

Malware Analysis

- Encryption algorithm can be something as simple as an XOR with the key or something more advanced
- Agents can contain the same or different keys, depending on how sophisticated they are
- It can fully or partially unpack, to make dumping harder. Key can be obtained in run time and not being persisted
- What are the challenges of developing malware using these constraints? How to do it?

Malware Analysis

- Is it easy to…
  - Develop a cryptographic machinery?
  - Include a BLOB with encrypted bytes in a binary?
  - Allocate memory with execution permissions?
  - Decrypt bytes, write them in allocated memory and jump to execute?
  - Develop code to be packed as a BLOB?

Malware Analysis

- Is it easy to…
  - Develop a cryptographic machinery? ✓
  - Include a BLOB with encrypted bytes in a binary? ✓
  - Allocate memory with execution permissions? ✓
  - Decrypt bytes, write them in allocated memory and jump to execute? ✓
  - Develop code to be packed as a BLOB? ✗

Malware Analysis

- Executable code (to be packed) has to be self-contained like shellcode
  - Why?
    - We don’t want unpacked code to be in the file system to evade anti-virus. Otherwise, we could have a binary generated by a compiler and normally load it
    - Thus, there is no “loader” that automatically resolves external libraries and do other tasks
    - Fully loading an ELF or a PE is complex to manually implement and would increase malware size

Malware Analysis

- We don’t want an IAT (Import Address Table) or a GOT (Global Offset Table) with functions that malware invokes because it would leak much information about its behavior
- We can statically link all external functions (glibc, in Linux) but would drastically increase size

Malware Analysis

- Data and code are part of the same bytes stream: there are no sections
- Code is PIC (Position-Independent-Code)
  - Instruction pointer relative addressing mode
  - Ready to execute no matter in which virtual address is allocated or unpacked

Malware Analysis

```
#include <stdio.h>
#include <sys/types.h>
#include <netinet/in.h>

extern int sockfd;

static int f ();

int main(){
    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    int i = f();
    ...
}
```

Malware Analysis

- It’s possible:
  - Include headers and use constants
  - Use local variables
  - Use static functions and variables
- It’s not possible:
  - Reference external global variables
  - Call external functions (dynamically link libraries)

Malware Analysis

- If we follow these rules and compile PIC, .text segment has “shellcode” bytes to pack
  - We can develop malware in C (instead of manually writing assembly) and it can be easily packed

Demo 4.1

Compile shellcode and pack malware

Malware Analysis

- Functions resolution
  - Calling OS APIs (instead of directly executing syscalls) has advantages: high level abstractions
    - How many syscalls would be needed to resolve a domain, or perform an HTTP request?
    - We could simply call getaddrinfo and curl APIs

Malware Analysis

- Functions resolution
  - How to resolve functions?
    - Linux
      - If libdl.so is mapped to the process:
      - resolve dlsym reading the library memory (based on ELF structure and base address available in /proc/<PID>/maps)
      - Use dlsym and dlopen to load libraries and resolve symbols
      - If dynamic loader is in the process, we can use it to resolve functions
      - We can implement our own resolver based on ELF structure and process libraries base addresses

Demo 4.2

Resolve dlsym and, with dlsym, getaddrinfo

Malware Analysis

- Functions resolution
  - Use libraries functions
    - Windows
      - Pointer to TIB (fs:0x30) → PEB
      - Iterate modules (dlls) loaded in the process until kernel32.dll is located
      - Resolve GetProcAddress browsing memory (PE structure)
      - Use LoadLibrary to load libraries and GetProcAddress to resolve symbols

Demo 4.3

Resolve kernel32.dll in Windows

Malware Analysis

- PEB – Process Environment Block
  - Undocumented structure, used by ntdll
  - It’s loaded with some information provided by the kernel
  - Located in user-space (thus, the process can read it)
  - Has global information about the process: list of loaded modules (PPEB_LDR_DATA), is the process being debugged?, session ID, parameters, etc.

Malware Analysis

```
typedef struct _PEB_LDR_DATA {
  BYTE       Reserved1[8];
  PVOID      Reserved2[3];
  LIST_ENTRY InMemoryOrderModuleList;
} PEB_LDR_DATA, *PPEB_LDR_DATA;

typedef struct _LDR_DATA_TABLE_ENTRY {
    ...
    PVOID DllBase;
    PVOID EntryPoint;
    PVOID Reserved3;
    UNICODE_STRING FullDllName;
    ...
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;
```

Malware Analysis

- Malware does not like debuggers nor sandboxes
  - IsDebuggerPresent (Windows API)
    - “BeingDebugged” field in PEB
  - Who is the parent process? Is there a debugger (TracerPid) in /proc/<PID>/status? (Linux)
  - Query physical resources (RAM, HDD, etc.). A system with few resources may not be real
  - Query drivers (Vendor IDs)
  - System uptime (GetTickCount Windows API)

Malware Analysis

- Malware does not like debuggers nor sandboxes
  - Benchmarking: how much time does it take to perform a costly computation?
  - Is there an emulated instruction? How much does it take to execute?
  - To measure time in x86/x86_64 the non-privileged RDTSC (read timestamp) instruction can be used
    - An approximate timer with a parallel thread decrementing a variable can be used

Malware Analysis

- Malware does not like debuggers nor sandboxes
  - If malware detects that it is being debugged, it may behave in a non-suspicious way
- Malware can remain in stand-by mode and operate only under specific conditions
- Thus, fully automated tools have drawbacks -particularly against the most advanced malware-. Manual analysis and malware patching to exhibit real behavior may be necessary (and fun!)

Malware Analysis

- Tools to monitor filesystems, network traffic or the registry may be complementary
  - However, if malware encrypts network communications or written files, debugging/instrumenting may be necessary to see data before encryption

Malware Analysis

- Fingerprinting limitations
  - Packing (with different keys)
    - It would be necessary to verify fingerprints after unpacking and evade anti-debugging techniques
  - Polymorphism
    - Suppose that malware needs to set EAX register to 0. What can it do?

Malware Analysis

- Suppose that malware needs to set EAX register to 0. What can it do?
  - XOR EAX, EAX
  - MOV EAX, $0
  - SUB EAX
  - Logic SHIFT
  - Etc.

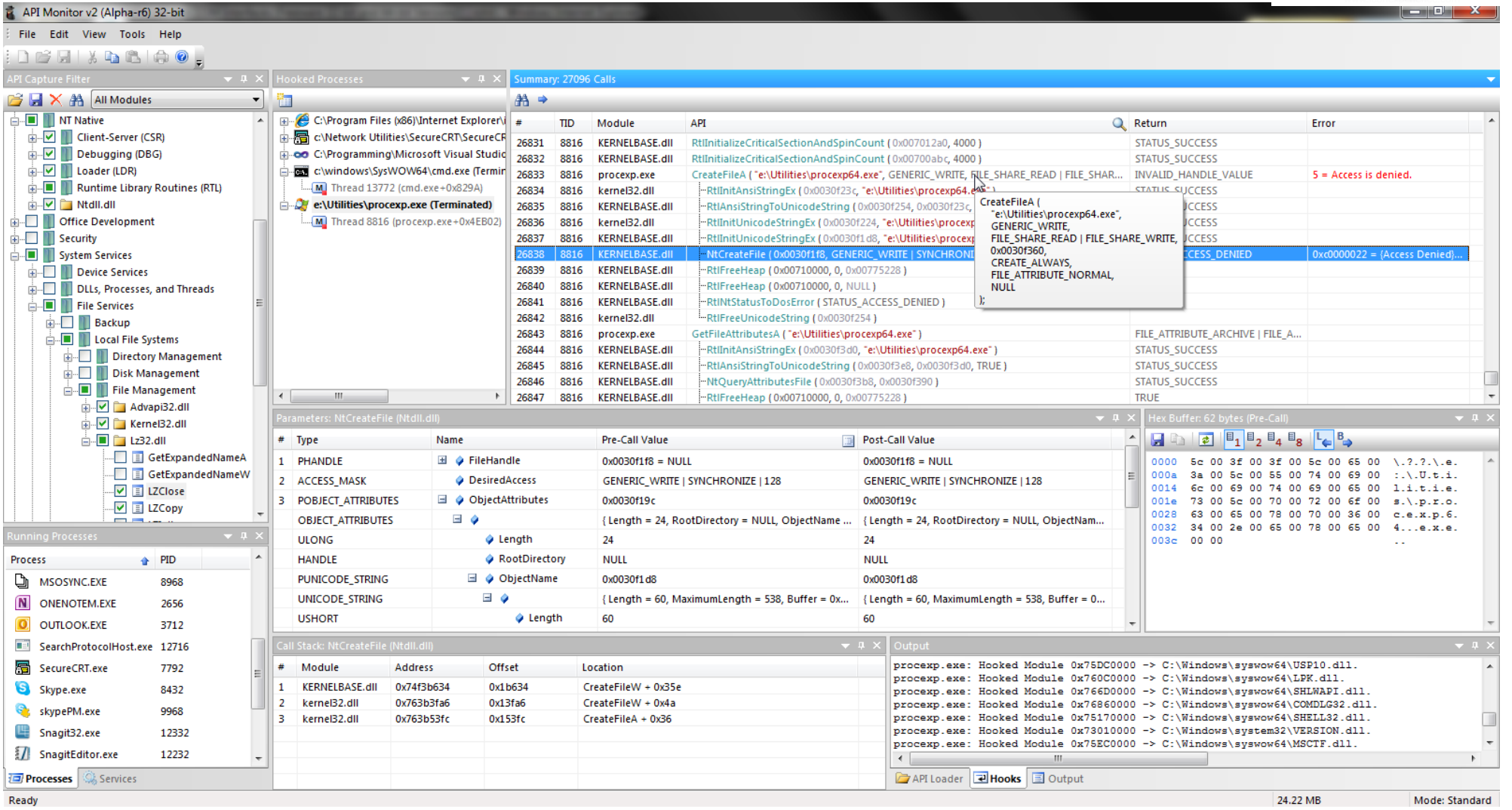
Malware Analysis

- Each instruction can be rewritten in different ways keeping the same semantic value
- Innocuous instructions can be added in-between (obfuscation)

Malware Analysis

- Other malware goals:
  - Worming → infect targets nearby
  - Privilege escalation - rootkit
  - Persistence
    - Init.d (Linux)
    - Service (Windows)
    - Windows Management Instrumentation (WMI)
    - Firmware / bootloader patching

Malware Analysis



Windows API Monitor

Lab 4.1

Analyze Lab 4.1 malware (ELF, x86) and describe its behavior

WARNING:

Execute inside a virtual machine only. Binary performs REAL damage. Usage of snapshots is recommended.

References

- https://www.virustotal.com
- https://github.com/ytisf/theZoo/tree/master/malwares