Reverse Engineering Class 4

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).



- 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



4

- 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 exfiltration techniques (intelligent and specific agents)
 - Avoid "noise" in both the endpoint and network Reverse Engineering | Class 4 | Martin Balao | martin.uy/reverse | v1.0 EN | CC BY-SA



- 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)



- 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"



- 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



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





- 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



- 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



- 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)



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





- 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



- 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



- 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



- 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



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





- 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



- 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



- 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



- 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



- 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





Binary malware





- 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



 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?





 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.)



- 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?



- 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?





- 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?



- 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



- 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



- 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





#include <stdio.h>



- 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)



- 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



- 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



- 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



- 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



- 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.



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 DIIBase; PVOID EntryPoint; PVOID Reserved3; UNICODE_STRING FullDIIName;

... } LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;



- 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 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 nonprivileged RDTSC (read timestamp) instruction can be used
 - An approximate timer with a parallel thread decrementing a variable can be used



- 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!)



- 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



- 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?





- 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.



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



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



👔 API Monitor v2 (Alpha-r6) 32-bit		Summer States of States	and the owner where the											- 0 - X
File Edit View Tools Help														
i D 62 9 X 🖬 🛍 🎝 💿 🖕														
API Capture Filter 🔷 🔻 🖡	×	Hooked Processes	→ ‡ ×	Summa	ry: 27096	5 Calls								-
🚰 🛃 🗙 🏦 🛯 All Modules	•			A 🏓										
		🕕 🏉 C:\Program File	#	TID	Module	API			Q	Return		Error	*	
🗄 🖳 🚺 Client-Server (CSR)		🖶 🔚 c:\Network Utili	ities\SecureCRT\SecureCl	26831	8816	KERNELBASE.dll	RtIInitializeCriticalSectionAndSpinC	Count	(0x007012a0, 4000)		STATUS	SUCCESS		
🖶 🗹 📗 Debugging (DBG)	🕂 👓 C:\Programmi		g\Microsoft Visual Studi	26832	8816	KERNELBASE.dll	RtlInitializeCriticalSectionAndSpin		ount (0x00700abc, 4000)		STATUS_SUCCESS			
🖶 🖳 🗾 Loader (LDR)		c:\windows\Sys	WOW64\cmd.exe (Termin	26833	8816	procexp.exe	CreateFileA ("e:\Utilities\procexp64	.exe",	GENERIC_WRITE, FILE_	SHARE_READ FILE_SHAR	INVALID	_HANDLE_VALUE	5 = Access is denied.	
🗈 🖳 📕 Runtime Library Routines (RTL)		M Thread 1377	72 (cmd.exe+0x829A)	26834	8816	kernel32.dll		"e:∖U	tilities\procexp64.eas	-t-FileA (STATUS	-success		
🕀 🖓 🧧 Ntdli.dli		e:\Utilities\proc	cexp.exe (Terminated)	26835	8816	KERNELBASE.dll	···RtIAnsiStringToUnicodeString (0x003	0f254, 0x0030f23c,	e:\Utilities\procexp64.exe",		CCESS		
Office Development		Imad 8816	(procexp.exe+0x4EB02)	26836	8816	kernel32.dll	RtlInitUnicodeStringEx (0x0030f	f224, *	e:\Utilities\procexp	SENERIC_WRITE,		JCCESS		
Security Security Security System Services				26837	8816	KERNELBASE.dll			d8, "e:\Utilities\procext FILE_SHARE_READ FILE_SHARE 0x0030f360.		RE_WRITE, ICCESS		0xc0000022 - (Access	Denied
				26839	8816	KERNELBASE.dll		00775	228.)	CREATE_ALWAYS,		CESS_DEMIED	Oxcooooo22 = {Access Defiled}	s Deffiedj
Device Services				26840	8816	KERNELBASE.dll		JLL)		-ILE_ATTRIBUTE_NORMAL,				
File Services	=			26841	8816	KERNELBASE.dll		ACCE	ESS_DENIED)					
				26842	8816	kernel32.dll	RtlFreeUnicodeString (0x0030f2	254)				_		
Local File Systems				26843	8816	procexp.exe	GetFileAttributesA ("e:\Utilities\pro	ocexp6	54.exe")		FILE_AT	TRIBUTE_ARCHIVE FILE_A		
Directory Management				26844	8816	KERNELBASE.dll		, ⊺e:\U	Itilities\procexp64.exe ⁺)	STATUS	SUCCESS		
⊕ □ Disk Management ⊕ □ □I File Management ⊕ □ □Advapi32.dll				26845	8816	KERNELBASE.dll	VELBASE.dll RtlAnsiStringToUnicodeString (0x0030f3e8, 0x0030f3d0, TRUE) NELBASE.dll -NtQueryAttributesFile (0x0030f3b8, 0x0030f390) VELBASE.dll -RtlFreeHeap (0x00710000, 0, 0x00775228)		0x0030f3e8, 0x0030f3d0, TRUE)		STATUS_SUCCESS			
				26846	8816	KERNELBASE.dll				STATUS	SUCCESS			
		•	20047	0010	KERINELDASE.dll	J// J/			IROE					
😥 🖓 📴 Kernel32.dll		Parameters: NtCreateFil	le (Ntdll.dll)								< Hex I	Buffer: 62 bytes (Pre-Call)	-	
Lz32.dll		# Type	Name	e 🖗 FileHandle		Pre-Call Value		Post-	st-Call Value		1 🖬 🖻 🛃		L ← B →	
		1 PHANDLE	🖽 🧼 FileHar			0x0030f1f8 = NULL			0x0030f1f8 = NULL GENERIC_WRITE SYNCHRONIZE 128		0000 5c 00 3f 00 3f	0 5c 00 3f 00 3f 00	00 5c 00 65 00 ∖.?.?.∖.e.	.?.\.e. ^
		2 ACCESS_MASK		dAccess		GENERIC_WRITE SYNCHRONIZE 128		GEN			000a 3a 00 5c 00 55 00) 74 00 69 00 :.\.U.t.i.	.U.t.i.
				Attributes		0x0030f19c		0x00	0x0030f19c		001	e 73 00 5c 00 70 00	72 00 6f 00 s.\.p.r.o.	.p.r.o.
						{ Length = 24, R	ootDirectory = NULL, ObjectName	{ Len	ength = 24, RootDirectory = NULL, ObjectNam		002	8 63 00 65 00 78 00 °	70 00 36 00 c.e.x.p.6.	.x.p.6.
Running Processes 🛛 👻 🕂 📿		ULONG 🔷 Le		ength	ngth 24				24		003	c 00 00		
Process PID MSOSYNC.EXE 8968 NONENOTEM.EXE 2656 OLITIOOK.EXE 3712		HANDLE	HANDLE PUNICODE_STRING UNICODE_STRING USHORT PART PART PART		tory	NULL		NUL	NULL					
		PUNICODE_STRING			ne	0x0030f1d8			0x0030f1d8					
		UNICODE_STRING				{ Length = 60, MaximumLength = 538, Buffer = 0x		{ Len	{ Length = 60, MaximumLength = 538, Buffer = 0					
		USHORT			Length 60				60 .					~
SearchProtocolHost.exe 12716	пľ	Call Stack: NtCreateFile	(Ntdll.dll)											→ ‡ ×
SecureCRT.exe 7792		# Module	Address Off	set	1	ocation			procexp.exe: Hos	oked Module 0x75DC000	0 -> C:	:\Windows\syswow64\USI	210.dll.	•
	=	KERNELBASE.dll 0x74f3b634 0x1b634 kernel32.dll 0x763b3fa6 0x13fa6		CreateFileW + 0x35e				procexp.exe: Hooked Module 0x760C000			:\Windows\syswow64\LPH	(.dll.		
Skype.exe 0452				3fa6	CreateFileW + 0x4a				procexp.exe: Hooked Module 0x766D000 procexp.exe: Hooked Module 0x7686000			:\Windows\syswow64\SH] :\Windows\syswow64\CO	WAFF.dll. (DLG32.dll.	
🍋 skypePM.exe 9968		3 kernel32.dll 0x763b53fc 0x153		53fc	C	reateFileA + 0x36		procexp.exe: Hooked Module 0x7517000			0 -> C:\Windows\syswow64\SHELL32.dll.			
Snagit32.exe 12332									procexp.exe: Hoo	oked Module 0x7301000	0 -> C:	\Windows\system32\VE	SION.dll.	
SnagitEditor.exe 12232									procexp.exe: Ho	okea Module Ux75EC000	u -≻ C:	:\windows\syswow64\MS	JF.dll.	
Processor Capiton										la alta	_			,
PIOCESSES Sal Dervices									AFILUQUE					

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/mal wares