SENTINELONE
THREAT INTELLIGENCE
IN-DEPTH

Ransomware Encryption Internals: A Behavioral Characterization

Antonio Cocomazzi
Threat Intelligence Researcher, SentinelOne

SentinelOne

whoami

- ➔ Threat Intelligence Researcher @ SentinelOne
- ➔ Mainly deal with malware analysis and reverse engineering
- ➔ Free time = coding offensive tools + deepin into Windows internals
- ➔ Previously presented at BlueHat, Black Hat, HITB, RomHack.

@splinter_code

@antonioCoco

Why this research

- ➔ Data encryption is the **core** functionality of every Ransomware and it enables their successful operations to extort money from the victims
- ➔ Static indicators are acceptable but **behavioral** indicators are gold
- ➔ Extracting Behavioral Indicators means deep knowledge -> lots of study -> very time **intensive**
- ➔ Providing a behavioral characterization should **ease** this --^
- ➔ Identifying behavioral **commonalities** can provide detection opportunities **generic** enough to identify all the most advanced Ransomware families, instead of relying of specific detection for specific families

Agenda

- ➔ Defining the data encryption scope
- ➔ Evolution, Trends and Unique features
- ➔ The behavioral characterization
- ➔ Behavioral detection based on overlapping implementations
    - Cross Drive File Enumeration detection
    - File Footer Writing detection
    - Encryption Key Randomization detection
    - Restart Manager API heavy usage detection
- ➔ Conclusion

Defining the data encryption scope

Defining the data encryption scope

- Data Encryption characterization requires a **dedicated** threat model wide enough to cover Ransomware behaviors in a **generic** way

- Four **Macro features**:
  - Files And Directories Enumeration
  - File Encryption
  - Encryption Parallelization
  - Encryption Optimization

- Selected Ransomware:
  - **Babuk**
  - **BlackMatter**
  - **Conti**
  - **Revil**

Some months later…

Evolution, Trends and Unique features

The shifts in the encryption schemes

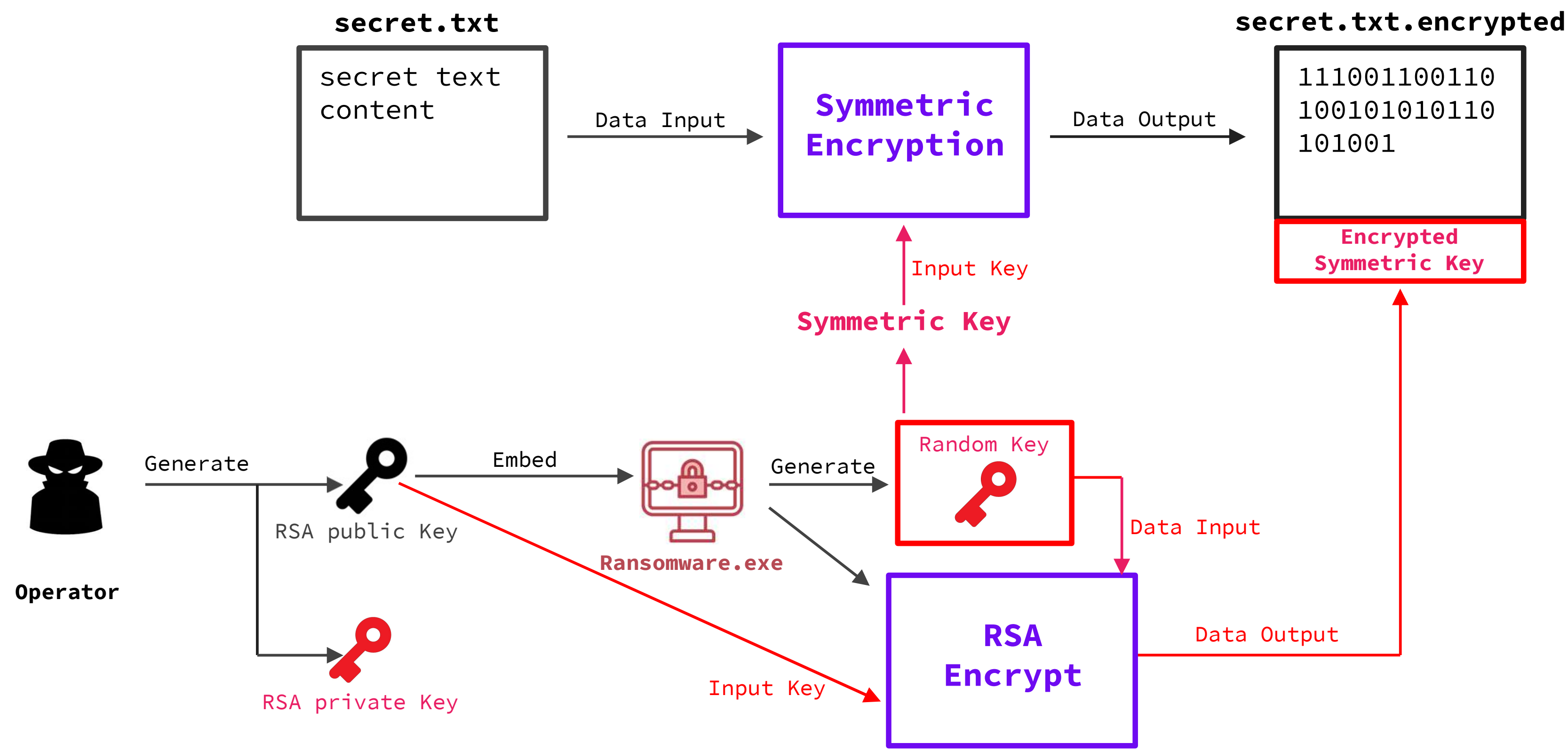
- Main shift is the adoption of **Elliptic-Curve Diffie-Hellman** (ECDH) key exchange algorithms instead of RSA as asymmetric encryption -> **main difference** the private key is never left on the victim host neither in encrypted form

- The evolution of the encryption implementation aims to avoid the usage of the **CryptoAPI** functionalities offered by the Windows operating system

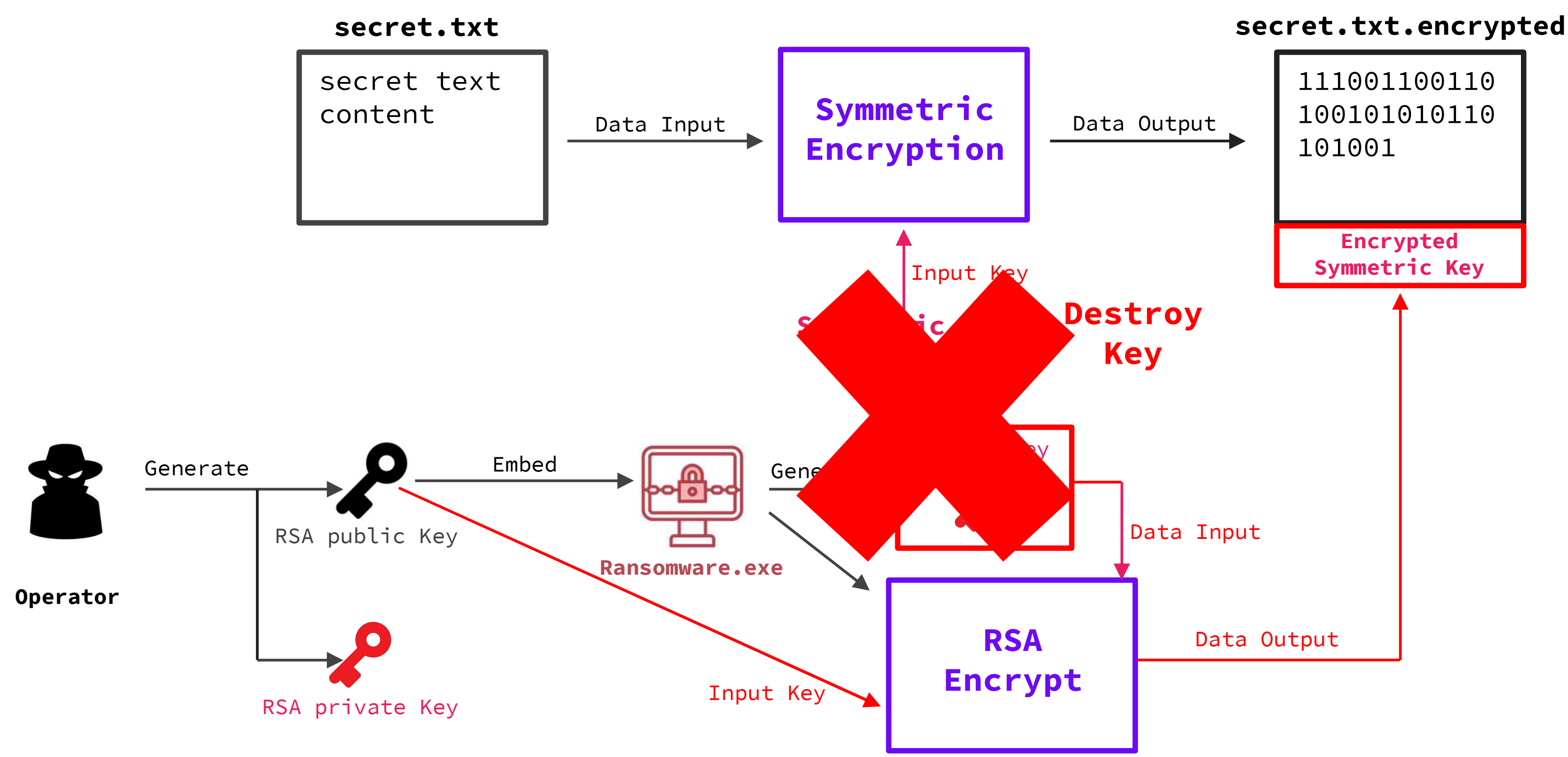
- Ransomware developers prefer to use **open-source** libraries or **custom implementation** for their symmetric and asymmetric encryption operations (e.g. curve25519-donna, HC-128, custom ChaCha20...)

- **All analyzed families append the information required to restore the symmetric private keys as a file footer!**

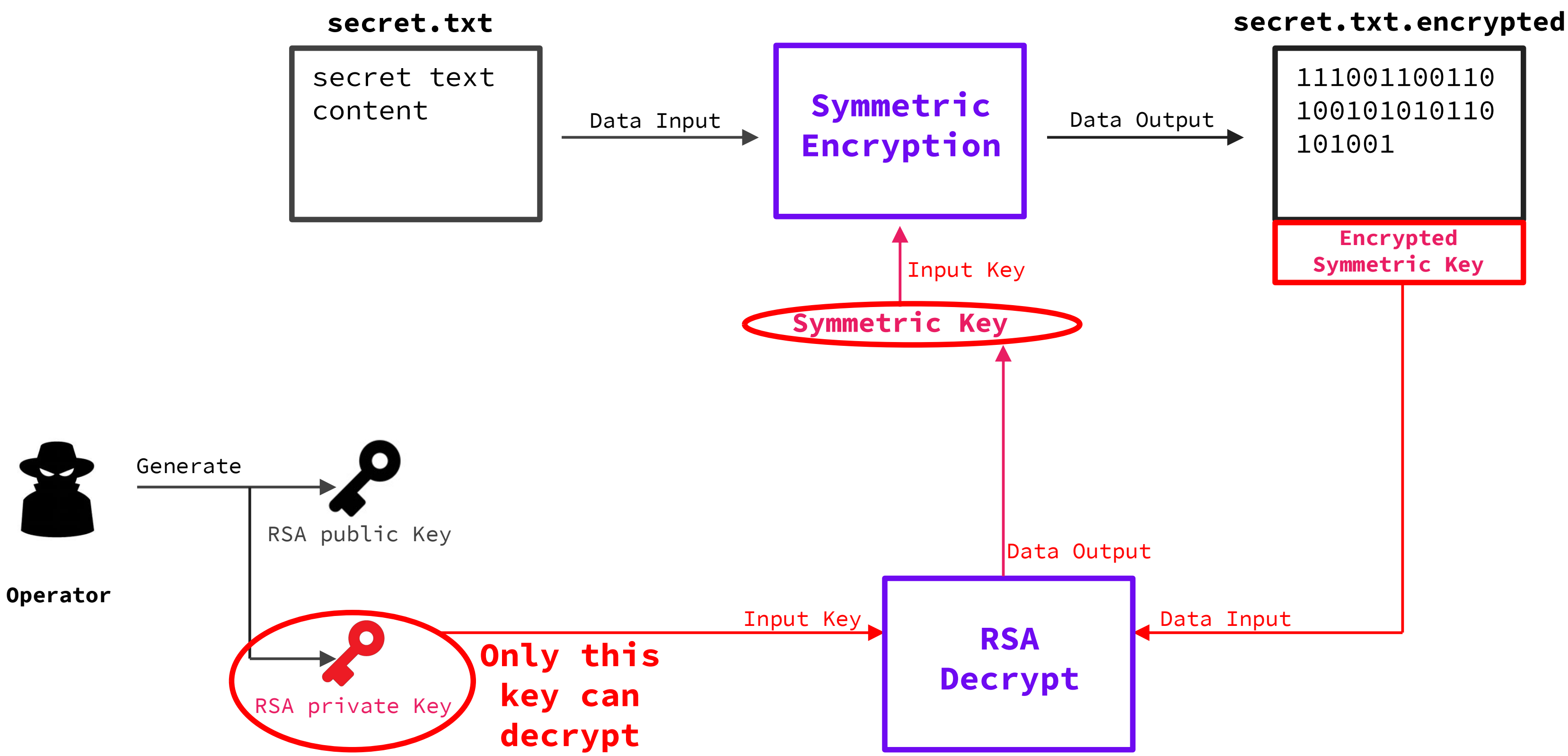
The shift from RSA to ECDH in Asymmetric Encryption: RSA



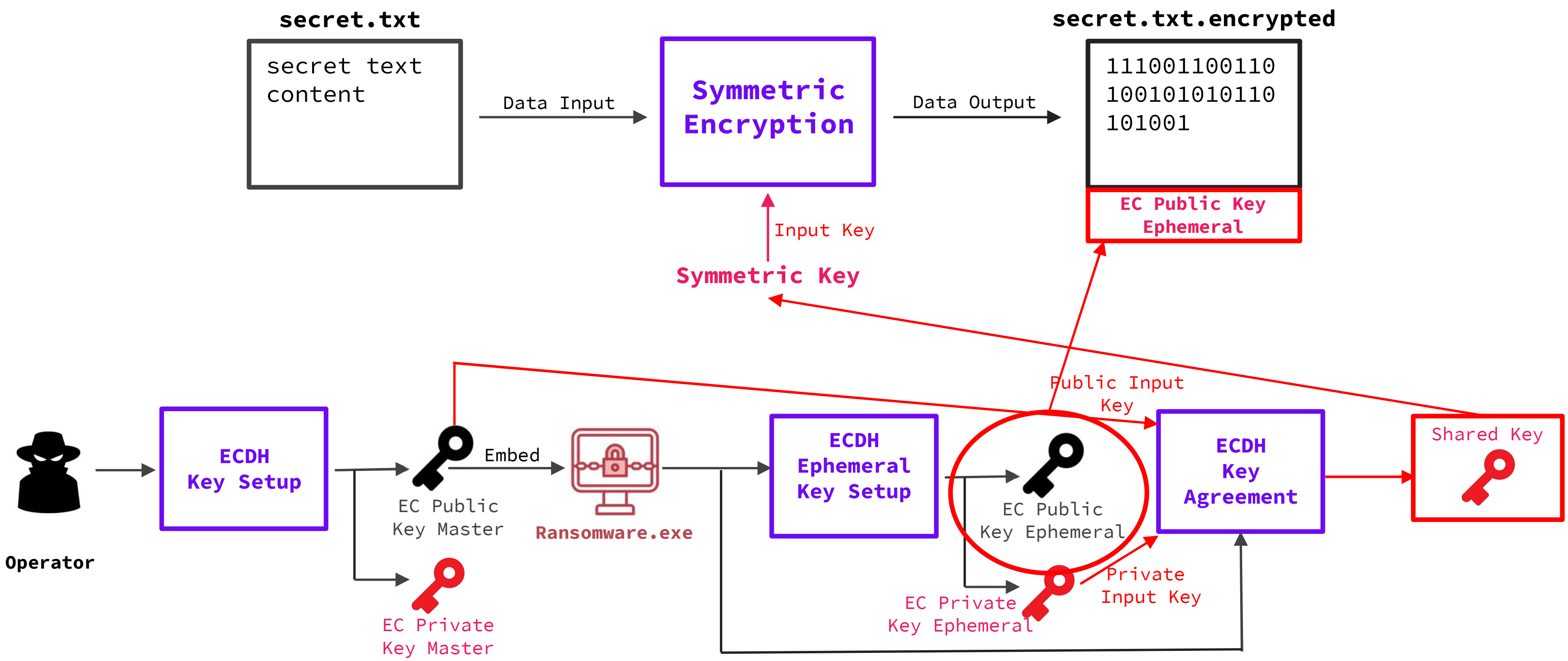
The shift from RSA to ECDH in Asymmetric Encryption: RSA



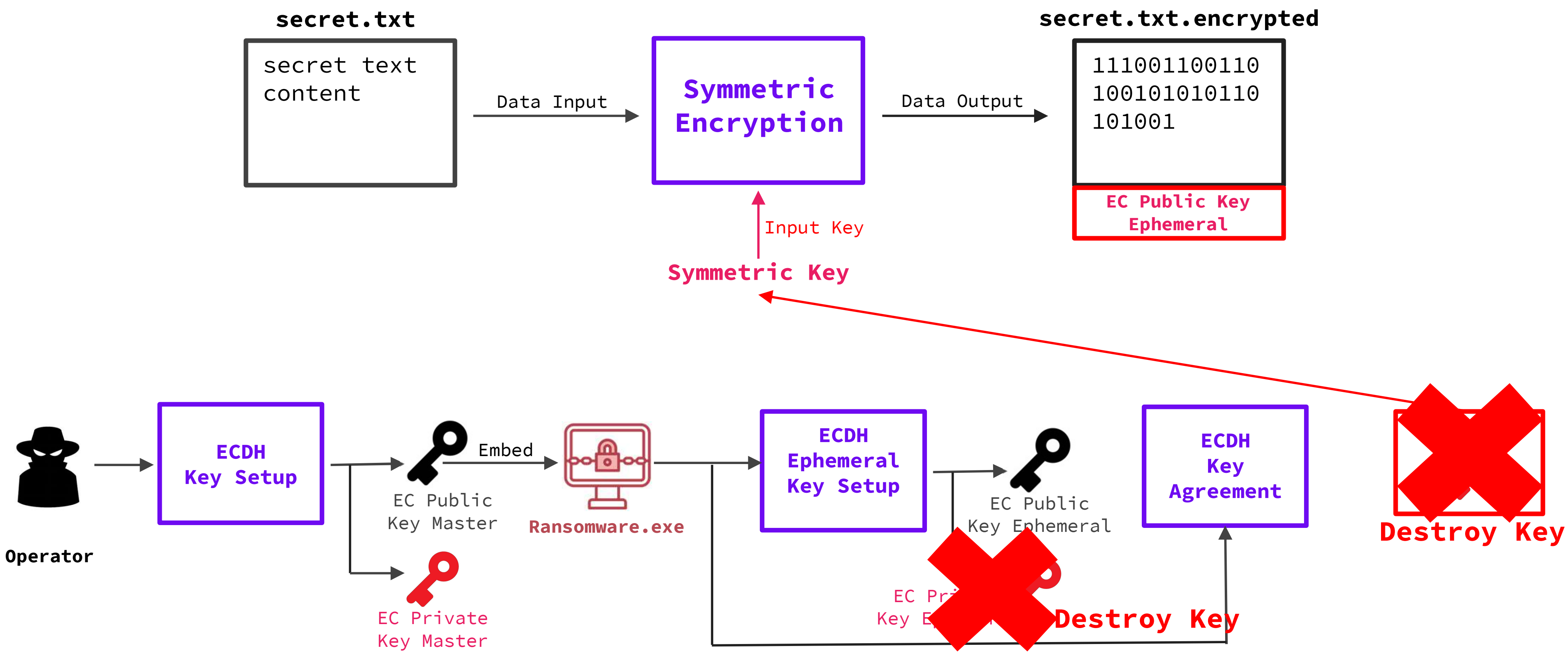
The shift from RSA to ECDH in Asymmetric Encryption: RSA



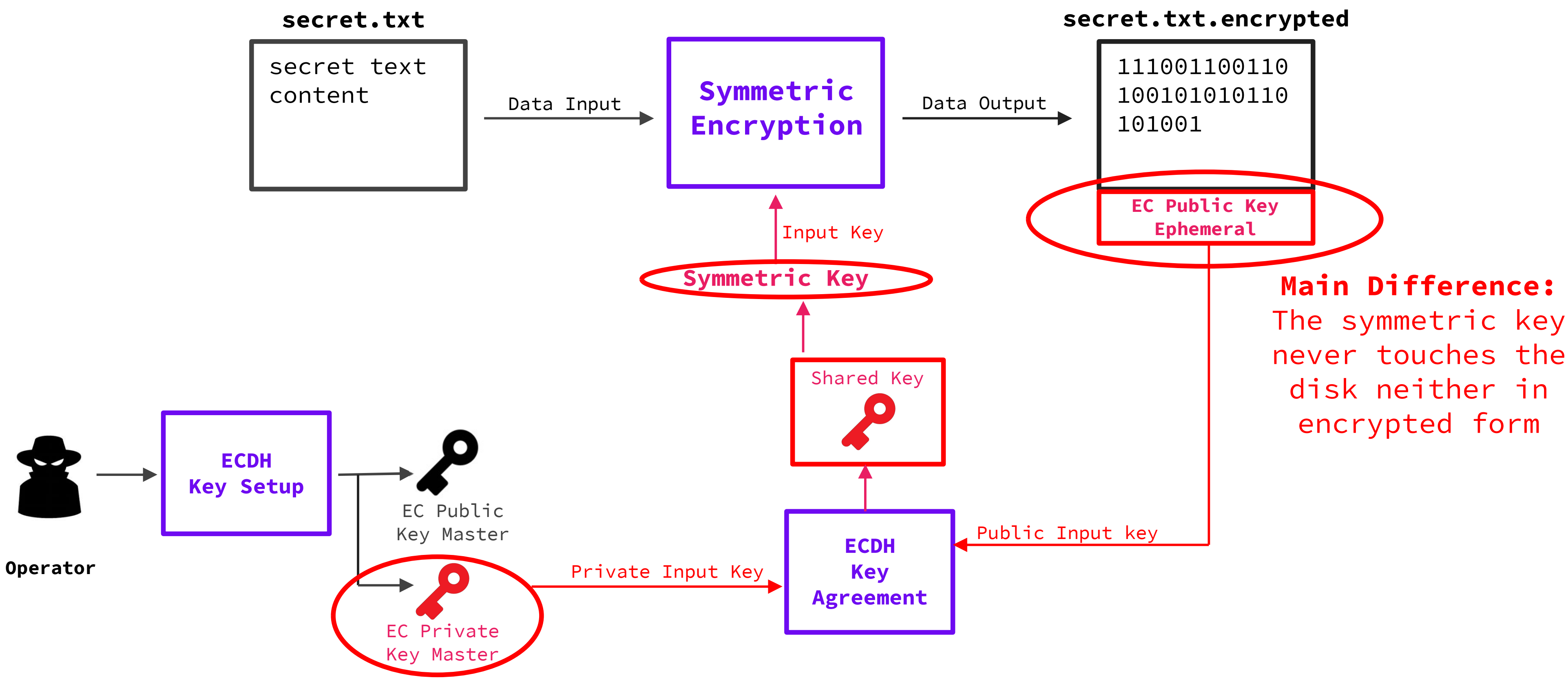
The shift from RSA to ECDH in Asymmetric Encryption: ECDH



The shift from RSA to ECDH in Asymmetric Encryption: ECDH



The shift from RSA to ECDH in Asymmetric Encryption: ECDH



Automated discovery of internal resources to target

- Every Ransomware implementation bundle **automated** ways to find and seek for relevant resources to encrypt

- The common trend identified is to enumerate all local directories and finding the remote shared resources

- Unique implementations that perform a more in-depth seek:
  - **BlackMatter** uses LDAP queries to retrieve all the computer names in the domain and build a list of the remote machines to encrypt files from
  - **Conti** retrieves the network addresses of the machines connected to the network through the ARP table stored locally

Automated discovery of internal resources to target

➔ Blackmatter automated LDAP discovery:

```
ADsOpenObject(“LDAP://rootDSE”, ... , IID_IADs, &IADs_object) ->

IADs_object::Get(..., &defaultNamingContext) ->

ADsOpenObject(wcscat(“LDAP://CN=Computers,”, defaultNamingContext.bstrVal, ...,
&IID_IADsContainer, &pADsContainer) ->

ADsBuildEnumerator(pADsContainer, &ppEnumVariant) ->

ADsEnumerateNext(ppEnumVariant, … , defaultNamingContext, ...) ->

IADs_object::Get(..., “dNSHostName”, &dnsHostNameVariant)
```

Growing focus in performance improvements

- ➔ One interesting evolution identified is the adoption of **tasks parallelization** in the Ransomware payloads -> The main motivation around that is to **shorten** the time of **reaction** of the security team behind the compromised organization
- ➔ All ransomware implementations analyzed prefer a native **multithreading** approach over a multiprocessing approach
- ➔ The main trends observed for the encryption parallelization is the usage of **I/O completion ports**

Growing focus in performance improvements

- Some unique performance improvements implementations...
- **Babuk** uses a unique approach with Semaphores and custom management of the thread pools and shared data structure.
  - Less overhead than using completion ports
- **BlackMatter** uses undocumented Windows functions to increase its process class and IO priority
  - This instructs the kernel to schedule primarily the execution of the threads running in the Ransomware process thus granting a performance improvement

Automated discovery of internal resources to target

➔ Blackmatter undocumented functions to increase process priority:

```
void __stdcall SetCurrentProcessPriority()
{
  PVOID ProcessIoPriorityHigh; // eax
  PVOID valuePtr; // ebx

  ProcessIoPriorityHigh = HeapAlloc_helper2(4);
  valuePtr = ProcessIoPriorityHigh;
  if ( ProcessIoPriorityHigh )
  {
    *ProcessIoPriorityHigh = IoPriorityHigh;
    NtSetInformationProcess(GetCurrentProcess(), ProcessIoPriority, ProcessIoPriorityHigh, 4u);
    *valuePtr <<= 9;
    NtSetInformationProcess(GetCurrentProcess(), ProcessPriorityClass, valuePtr, 2u);// PROCESS_PRIORITY_CLASS_ABOVE_NORMAL (6)
    *valuePtr = 7;
    NtSetInformationProcess(GetCurrentProcess(), ProcessDefaultHardErrorMode, valuePtr, 4u);
    HeapFree_helper(valuePtr);
  }
}
```

Additional efforts to maximize the encryption damages

- ➔ Ransomware developers ensure that the disruptive operations carried out by their Encryptor have a **higher** impact on the targeted systems
- ➔ The common trend is to **kill** a set of **processes** and **services** starting from a list of “unwanted” names
- ➔ Moreover, for unknown processes that hold lock conditions on files, the **Restart Manager API** are used to identify all the processes that prevent the successful encryption of files already in use

Additional efforts to maximize the encryption damages

- Another common feature is the usage of functions to erase volume backups (i.e. shadow copies)

- The methods observed:
  - **Vssadmin.exe (delete shadows, resize shadowstorage)**
    Utility to delete or resize the shadow copies
  - **Using COM (IWbemLocator, IWbemContext, IWbemServices)**
    Out-of-process COM objects to interact with the VSS providers through WMI services

Automated discovery of internal resources to target

➔ Babuk implementation for killing file lock holders:

```
if (dwError = RmStartSession(&dwSession, 0, szSessionKey) == ERROR_SUCCESS) {
    if (dwError = RmRegisterResources(dwSession, 1, (LPCWSTR*)&filePath, 0, NULL, 0, NULL) == ERROR_SUCCESS) {
        DWORD dwReason;
        UINT nProcInfoNeeded;
        UINT nProcInfo = 10;
        RM_PROCESS_INFO rgpi[10];
        if (dwError = RmGetList(dwSession, &nProcInfoNeeded, &nProcInfo, rgpi, &dwReason) == ERROR_SUCCESS) {
            for (UINT i = 0; i < nProcInfo; i++) {
                if (rgpi[i].ApplicationType != RmExplorer && rgpi[i].ApplicationType != RmCritical && GetCurrentProcessId() != rgpi[i
                    HANDLE hProcess = OpenProcess(SYNCHRONIZE | PROCESS_TERMINATE, 0, rgpi[i].Process.dwProcessId);
                    if (hProcess != INVALID_HANDLE_VALUE) {
                        TerminateProcess(hProcess, 0);
                        WaitForSingleObject(hProcess, 5000);

                        CloseHandle(hProcess);
                    }
```

The behavioral characterization

The behavioral characterization

Feature

→ **Files And Directories Enumeration**

- Mount hidden volumes
- Local Drive Enumeration
- Remote Drive Enumeration
- File Enumeration

Sub-features

Feature

→ **File Encryption**

- Asymmetric Encryption
- Symmetric Encryption
- Key Randomization
- Encrypted Block Writing
- File Footer Writing

Feature

→ **Encryption Optimization**

- Kill unwanted Services
- Kill unwanted Processes
- Shadow Copies Deletion
- Kill file lock holders
- Increase process priority

Sub-features

Feature

→ **Encryption Parallelization**

- Multi threading
- Synchronization

The behavioral characterization

- The various Ransomware families analyzed implements the sub-features in various ways
- By collecting all the details about the implementations it’s possible to **map** the **implementations** of each sub-features to the corresponding family
- The mapping has been based on the **NT/Win32 API** usage of the implementations
- The **goal** of this mapping is to provide a way to recognize overlapping implementations across families and ease the development of effective detection to identify Ransomware behaviors commonalities

The behavioral characterization

➔ Results for “Files And Directories Enumeration”:

Sub-Feature	Implementation	Babuk	BlackMatter	Conti	Revil
Mount hidden volumes	GetDriveType	√			
	GetLogicalDriveStrings		√		
	FindFirstVolume	√	√		
	GetVolumePathNamesForVolumeName	√	√		
	DeviceIoControl (IOCTL_DISK_GET_PARTITION_INFO_EX)		√		
	FindNextVolume	√	√		
	SetVolumeMountPoint	√	√		
Local Drive Enumeration	GetDriveType	√	√	√	√
	GetLogicalDrive	√			
	GetLogicalDriveStrings		√	√	
	NetShareAdd(NULL)				√

Sub-Feature	Implementation	Babuk	BlackMatter	Conti	Revil
Remote Drive Enumeration	WNetGetConnection	√			
	NetShareEnum	√	√	√	√
	WNetOpenEnum				√
	WNetEnumResource				√
	GetIpNetTable			√	
	DsGetDcName		√		
	DsGetDcOpen		√		
	DsGetDcNext		√		
	ADsOpenObject		√		
	ADsBuildEnumerator		√		
	ADsEnumerateNext		√		
File Enumeration	FindFirstFile	√		√	√
	FindFirstFileEx	√	√	√	√
	FindNextFile	√	√	√	√

Public link of the results → https://docs.google.com/spreadsheets/d/1PprkVGsNYFQ39yfqobiBpIg0qhfXz3__XQscqR7Gv9I/edit?usp=sharing

The behavioral characterization

➔ Results for “Files Encryption”:

Sub-Feature	Implementation	Babuk	BlackMatter	Conti	Revil
Asymmetric Encryption	RSA 1024-bit key len. custom impl.		√		
	curve25519 128-bit key len. open source impl.	√			√
	RSA 4096-bit key len. CryptoAPI impl.			√	
Symmetric Encryption	HC-128 256-bit key len. open source impl.	√			
	ChaCha20 128-bit key len. custom impl.		√		
	AES-256-CBC 256-bit key len. CryptoAPI impl.			√	
	Salsa20 256-bit key len. open source impl.				√

Sub-Feature	Implementation	Babuk	BlackMatter	Conti	Revil
Key Randomization	CryptGenRandom	√			√
	CryptGenKey			√	
	RDRAND		√		√
	RDSEED		√		
	RDTSC				√
Encrypted Block Writing	MoveFileEx	√	√	√	√
	CreateFile	√	√	√	√
	ReadFile	√	√	√	√
	SetFilePointerEx	√	√	√	√
	WriteFile	√	√	√	√
File Footer Writing	SetFilePointerEx(FILE_END)	√	√	√	√
	WriteFile	√	√	√	√

Public link of the results → https://docs.google.com/spreadsheets/d/1PprkVGsNYFQ39yfqobiBpIg0qhfXz3__XQscqR7Gv9I/edit?usp=sharing

The behavioral characterization

➔ Results for “Encryption Optimization”:

Sub-Feature	Implementation	Babuk	BlackMatter	Conti	Revil
Kill unwanted Services	CreateProcess(cmd /c net stop)			√	
	OpenSCManager	√	√		√
	OpenService	√	√		√
	QueryServiceStatusEx(SC_STATUS_PROCESS_INFO)	√			
	EnumServicesStatusEx(SC_ENUM_PROCESS_INFO)		√		√
	EnumDependentServices	√			
	ControlService(SERVICE_CONTROL_STOP)	√	√		√
	DeleteService		√		√
	CoCreateInstance (IWbemServices)				√
Kill unwanted processes	CreateToolhelp32Snapshot	√		√	√
	NtQuerySystemInformation(SystemProcessInformation)	√	√	√	√
	Process32First	√		√	√
	Process32Next	√		√	√
	OpenProcess	√	√	√	√
	NtOpenProcess	√	√	√	√
	TerminateProcess	√		√	√
	NtTerminateProcess	√	√	√	√
	CoCreateInstance (IWbemServices)				√

Sub-Feature	Implementation	Babuk	BlackMatter	Conti	Revil
Shadow Copies Deletion	ShellExec (vssadmin.exe)	√			
	CreateProcess(vssadmin.exe)			√	
	CoCreateInstance (IWbemServices)		√		√
Kill file lock holders	RmStartSession	√	√	√	√
	RmRegisterResource	√	√	√	√
	RmGetList	√	√	√	√
	RmShutdown			√	
	TerminateProcess	√			√
	NtTerminateProcess	√	√		√
	ControlService(SERVICE_CONTROL_STOP)		√		√
	DeleteService		√		√
Increase process priority	NtSetInformationProcess(ProcessIoPriority)		√		
	SetPriorityClass				√
	NtSetInformationProcess(ProcessPriorityClass)		√		√

The behavioral characterization

➔ Results for “Encryption Parallelization”:

Sub-Feature	Implementation	Babuk	BlackMatter	Conti	Revil
Multi threading	GetSystemInfo	√		√	√
	NtCurrentPeb()->NumberOfPr ocessors	√	√	√	√
	CreateThread	√	√	√	√
	WaitForMultipleObject	√	√	√	√
Synchronization	CreateSemaphore	√			
	WaitForSingleObject	√			
	CreateIoCompletionPort		√	√	√
	PostQueuedCompletionStatus		√	√	√
	GetQueuedCompletionStatus		√	√	√

Public link of the results → https://docs.google.com/spreadsheets/d/1PprkVGsNYFQ39yfqobiBpIg0qhfXz3__XQscqR7Gv9I/edit?usp=sharing

Behavioral detection based on overlapping implementations

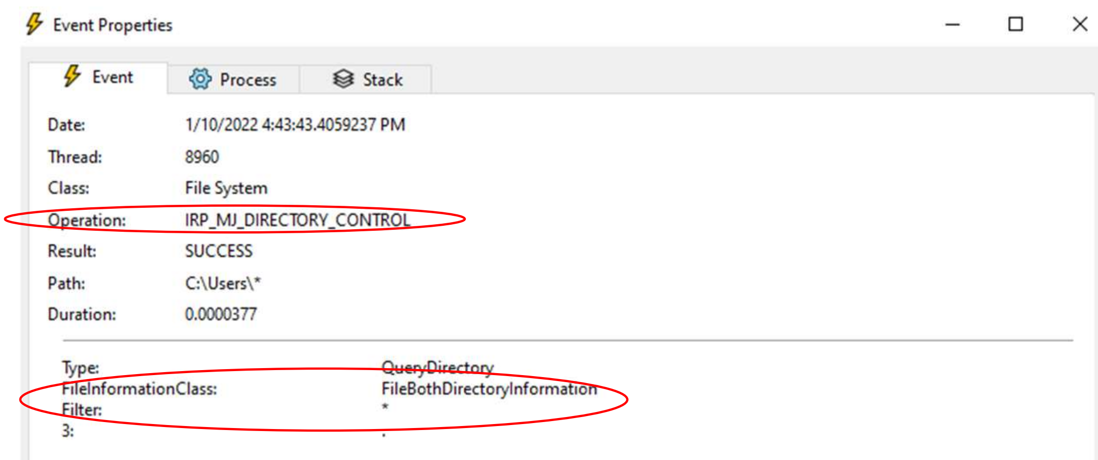
Behavioral detection based on overlapping implementations

➔ Overlapping sub-features implementations:

Feature	Sub-Feature	Implementation	Babuk	BlackMatter	Conti	Revil
Files And Directories Enumeration	File Enumeration	FindFirstFileEx	√	√	√	√
		FindNextFile	√	√	√	√
File Encryption	Key Randomization	CryptGenRandom	√			√
		CryptGenKey			√	
	File Footer Writing	SetFilePointerEx(FILE_END)	√	√	√	√
		WriteFile	√	√	√	√
Encryption Optimizations	Kill file lock holders	RmStartSession	√	√	√	√
		RmRegisterResource	√	√	√	√
		RmGetList	√	√	√	√

Cross Drive File Enumeration detection

- Every Ransomware analyzed performs the sub-feature “File Enumeration” with the same implementation:
  - **FindFirstFileEx**(“[DRIVE]:\[PATH]*”, ...)
  - **FindNextFile()**
- The usage of the Win32 Api function **FindFirstFileEx()** combined with the wildcard ‘*’ char appended at the end of each path found on the system does generate a specific **IRP** at the kernel level:



Cross Drive File Enumeration detection

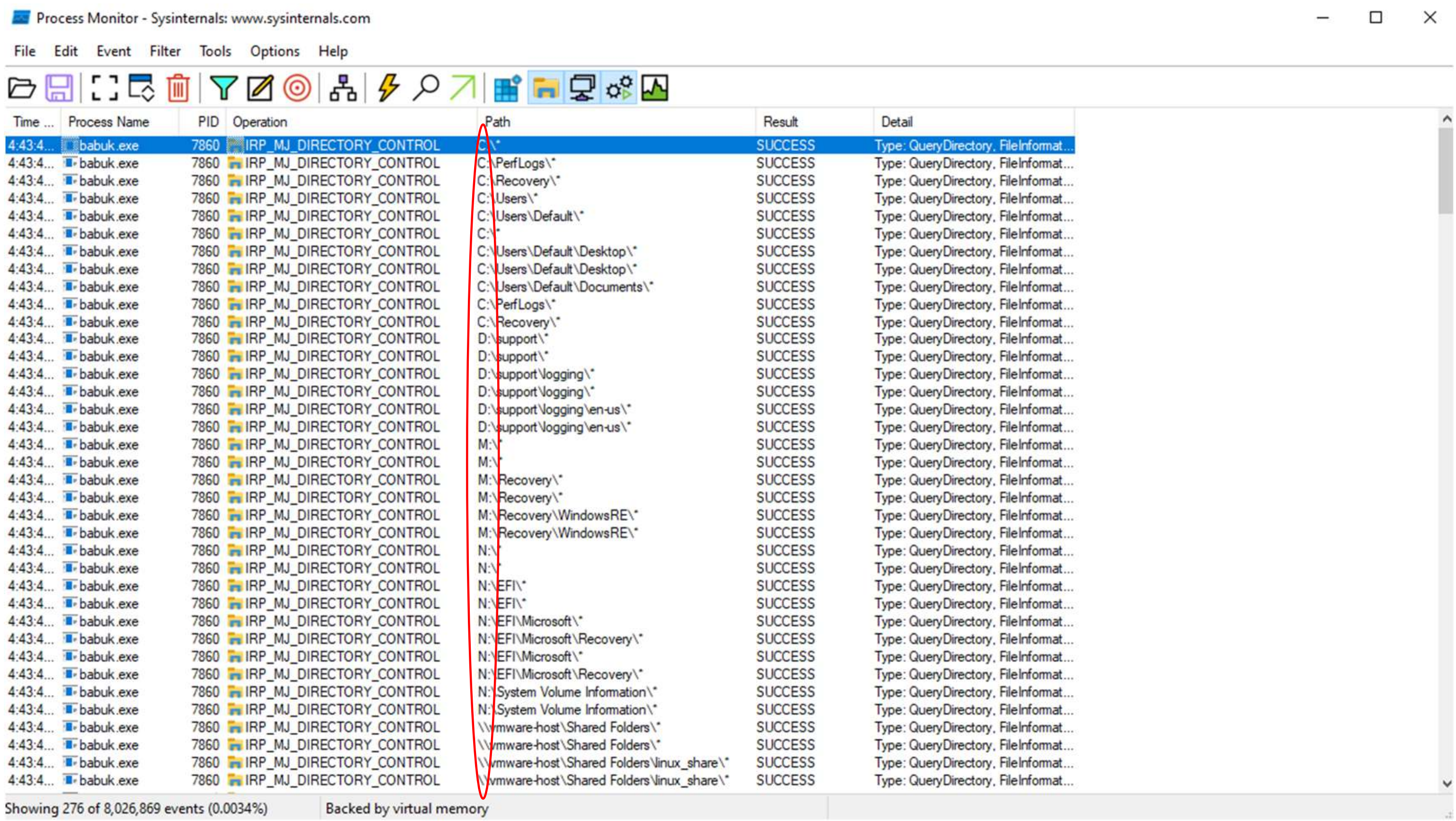
- A potential problem with this approach is that it could be prone to a high false positive rate

- Here is where it comes into play the concept of the “Cross Drive” file enumeration.
    - Every Ransomware performs a series of operations to identify all the hidden, local and remote drives on the system prior to the file enumeration operation

- The **IRP_MJ_DIRECTORY_CONTROL IRP** is dispatched to multiple logical drives. This makes the operation quite unique and abnormal for usual benign applications

- The detection spot occurs at the **kernel** level :)

Cross Drive File Enumeration detection


Process Monitor - Sysinternals: www.sysinternals.com

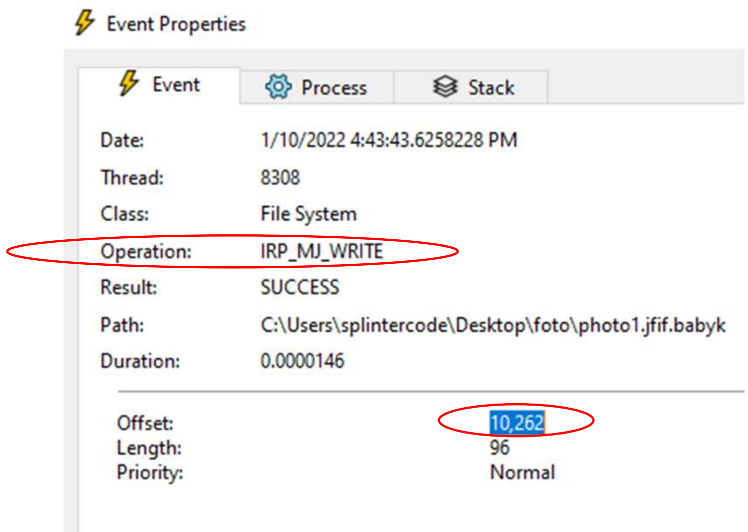
File Edit Event Filter Tools Options Help

Time ...	Process Name	PID	Operation	Path	Result	Detail
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	C:*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	C:\PerfLogs*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	C:\Recovery*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	C:\Users*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	C:\Users\Default*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	C:*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	C:\Users\Default\Desktop*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	C:\Users\Default\Desktop*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	C:\Users\Default\Documents*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	C:\PerfLogs*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	C:\Recovery*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	D:\support*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	D:\support*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	D:\support\logging*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	D:\support\logging*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	D:\support\logging\en-us*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	D:\support\logging\en-us*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	M:*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	M:*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	M:\Recovery*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	M:\Recovery*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	M:\Recovery\WindowsRE*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	M:\Recovery\WindowsRE*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	N:*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	N:*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	N:\EFI*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	N:\EFI*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	N:\EFI\Microsoft*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	N:\EFI\Microsoft\Recovery*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	N:\EFI\Microsoft*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	N:\EFI\Microsoft\Recovery*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	N:\System Volume Information*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	N:\System Volume Information*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	\\vmware-host\Shared Folders*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	\\vmware-host\Shared Folders*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	\\vmware-host\Shared Folders\linux_share*	SUCCESS	Type: QueryDirectory, FileInformat...
4:43:4...	babuk.exe	7860	IRP_MJ_DIRECTORY_CONTROL	\\vmware-host\Shared Folders\linux_share*	SUCCESS	Type: QueryDirectory, FileInformat...

Showing 276 of 8,026,869 events (0.0034%) Backed by virtual memory

File Footer Writing detection

- Every Ransomware analyzed performs the sub-feature “File Footer Writing” with the same implementation:
  - **SetFilePointerEx**(hFile, …, FILE_END)
  - **WriteFile**(hFile, fileFooterStruct, sizeof(fileFooterStruct), …)
- The combination of these Win32 Api functions generate a specific IRP with specific characteristics at the **kernel** level:

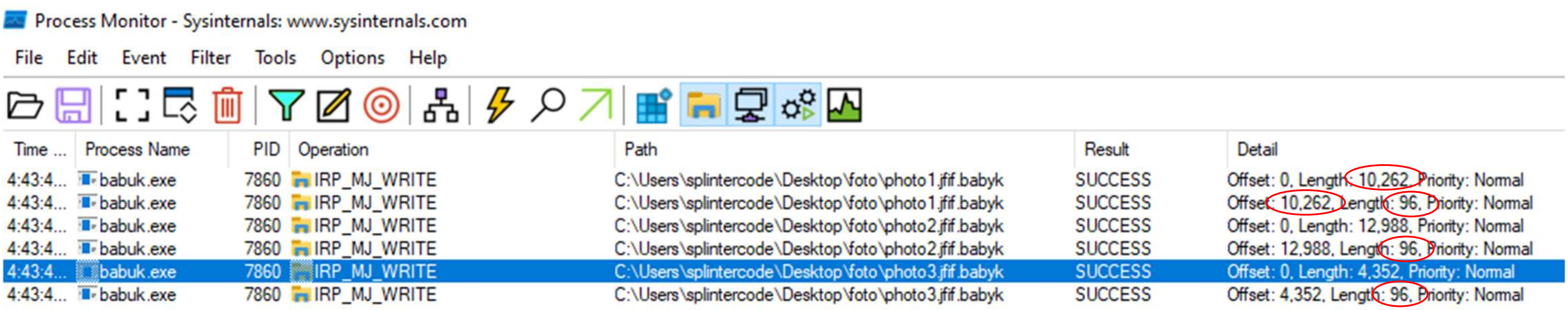


File Footer Writing detection

- In a pre operation callback **IRM_MJ_WRITE**, if the parameter **IrpSp->Parameters.Write.ByteOffset** is equal to the actual size of the file in which the write is happening
  - It means that's an **append** operation
  - Then the value **IrpSp->Parameters.Write.Length** should be stored for further validation
  - This value represents the actual **size of the struct** used by the ransomware to append the footer information needed for the decryption
- Unfortunately, the file footer struct size **differs** between Ransomware implementations
  - We can aggregate the number of append operations that have the same **recurring** length
  - This characterizes the behavior of a Ransomware trying to write its own **file footer** to each file it encrypts

File Footer Writing detection

➔ **Babuk** example of "**File Footer Writing**" implementation:



File Footer Writing detection

- ➔ By monitoring the file writes performed in this way, it is possible to **count** how many **file markers** are appended to files

- ➔ E.g. We can keep track of these file writes with a **dictionary data structure** where on the **key** is stored the **Length** of the write operation and as a **value** the **counter** of how many times that write with that size has been appended to a file

- ➔ When a Ransomware is executed it should be observed that the counter contained in the value of a specific key of the dict is **exceeding** a **threshold**

- ➔ The detection spot occurs at **kernel** level :)

Restart Manager API heavy usage detection

- Restart Manager API usage common implementation:
  - **RmStartSession()**
  - **RmRegisterResource**(..., &filePath, …)
  - **RmGetList()**
- Whenever a call to **CreateFile()** fails to return a valid file handle, the Ransomware assumes the failure is due to some file locking mechanism held by some process
  - This generates a heavy usage of the Restart Manager APIs
- Lowest NT API to monitor by reversing **RmGetList()** from **RstrtMgr.dll**:
  - RmGetList()
  - CRestartManager::GetAffectedApplications()
  - CRestartManager::UpdateInternalData()
  - RmFileFactory::UniqueAffectedPids()
  - RMRegisteredFile::AffectedPids()
  - **NtQueryInformationFile()**

Restart Manager API heavy usage detection

- The invocation of **NtQueryInformationFile()** from RmGetList() uses an undocumented **FILE_INFORMATION_CLASS** value of **FileProcessIdsUsingFileInformation**

- Peak usage of this call performed with the **FileProcessIdsUsingFileInformation** value (0x2F) could be used to characterize the usage of the Restart Manager API specifically by a Ransomware

- The detection spot occurs at **userland** level :(

Encryption Key Randomization detection

- Private keys are generated through **PRNG** (pseudo random number generator) either for symmetric or asymmetric encryption
  - This randomization operation is performed for each file encrypted thus generating a **high volume** usage of the PRNG functionalities
  - These implementations rely on the Win32 API calls **CryptGenRandom()** and **CryptGenKey()** from **advapi32.dll**
  - The observed value **“dwLen”** for the **CryptGenRandom()** call do overlap between the different implementations
  - Usually this value is equal to 16 or 32 that match the size of the private keys for the encryption algorithms implemented (so 128 or 256 bits)

- Not very generic and robust like others detection methods...
  - But it can be used as an **opportunistic** way to detect implementation based on these APIs usage

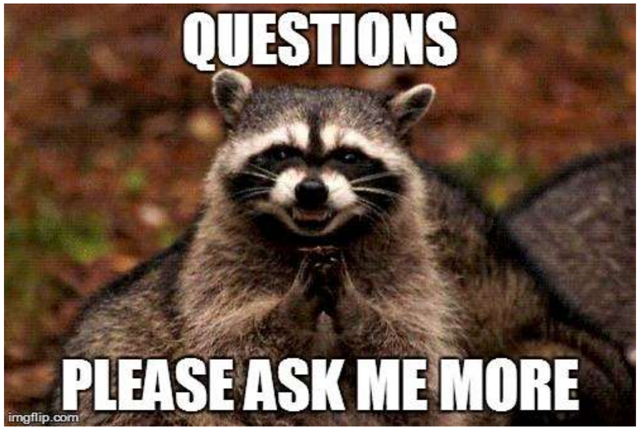
- The detection spot occurs at **userland** level :(

Conclusion

- ➔ Giving **insights** of what are the core operations characterizing the data encryption stage makes **analysis** of these complex threats **easier**
- ➔ Identifying commonalities in implementations allows to create **behavioral indicators** based on the side-effects generated by those operations valid for most Ransomware families
- ➔ The main reason for preferring behavioral indicators over static indicators is because they are much more reliable and **harder to evade**
- ➔ **TL;DR** Behavioral detection is the right approach for **scalable** Ransomware **countermeasures**

Special Thanks

- ➔ SentinelOne's team (Claudia, Daniel, Andreas)
- ➔ Idan Weizman
- ➔ Chuong Dong (@cPeterr)



Thank You!

@splinter_code

splintercod3@gmail.com