Encrypted Chaos: Analysis of Crytox Ransomware

Published: 2023-06-01 · Archived: 2026-04-05 23:04:28 UTC

Crytox Ransomware is a 64 bit executable, developed in C and usually deployed by packing the compiled executable with UPX. On unpacking, the size of the payload is around 2.9 MB, which is unusually high for a malware. On analyzing the binary we came to know that an entire uTox client was embedded at the start of the .text section.

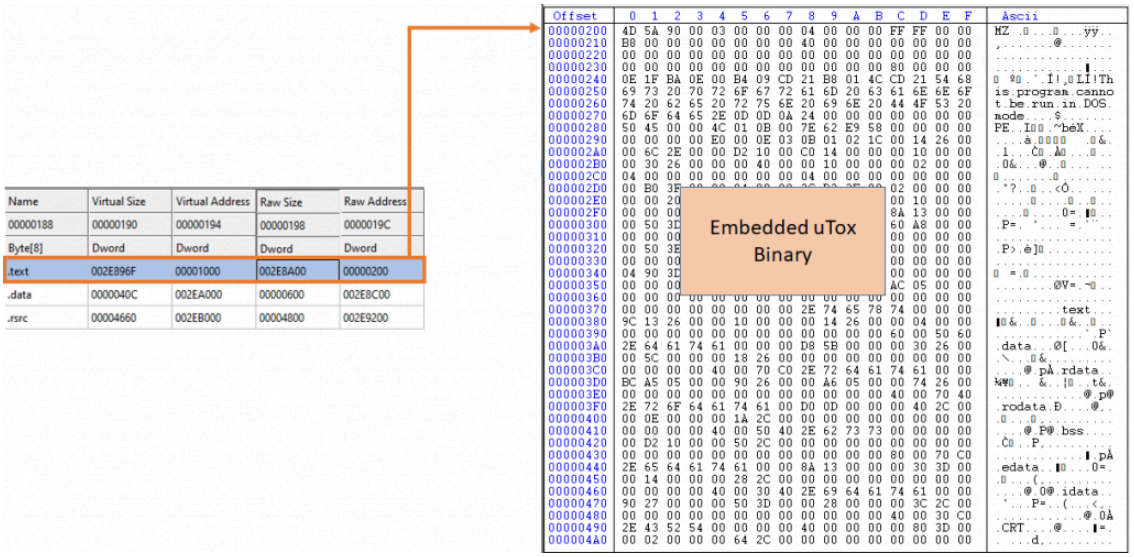

Figure 1: Embedded uTox binary

On execution, the ransomware decrypts a configuration file using AES algorithm, drops the uTox application in the path mentioned in the configuration file and injects a shellcode into a native Windows process mentioned in the configuration. This shellcode deletes the volume shadow copies and then injects a new shellcode into another native process which runs with a specific cmdline argument (in our case svchost with netsvcs cmdline was targeted). The final injected shellcode is responsible for encrypting the user files on disk with a ".waiting" extension.

Analysis

Stage – 1

API Resolving

Win32 APIs are dynamically resolved at runtime, it uses ROR7 for calculating module/DLL name hash, and ROR5 for calculating the export API hash. The binary contains hardcoded values which are the sum of module hash and API hash it needs to resolve and call, the equivalent code converted to python is shown below.

```
"""""""""""""""""""""""""
Crytox API Resolving
```

```
"""""""""""""""""""""""""

# https://www.geeksforgeeks.org/rotate-bits-of-an-integer/
def rightRotate(n, d):
    return (n >> d) | (n << (INT_BITS - d)) & 0xFFFFFFFF


def calculateHash(moduleName, moduleAPIList):
    moduleName = moduleName.upper()
    moduleName_bytes = moduleName.encode("utf-16le") + b'\x00\x00'
    moduleHash = 0
    for byte in moduleName_bytes:
        val = ord(chr(int(byte)))
        moduleHash = (val + ((rightRotate(moduleHash, 7)) & 0xFFFFFFFF)) & 0xFFFFFFFF
    for api in moduleAPIList:
        api_bytes = api.encode("utf-8") + b'\x00'
        apiHash = 0
        for byte in api_bytes:
            val = ord(chr(int(byte)))
            apiHash = (val + ((rightRotate(apiHash, 5)) & 0xFFFFFFFF)) & 0xFFFFFFFF
        exp = hex((moduleHash + apiHash) & 0xFFFFFFFF)
        if int(exp, 0) in Hash_present_in_Binary:
            print(f"{api} = {exp}")
```

Complete List of API's that are resolved by the file, can be seen in Appendix A.

Stage – 1 Configuration and Key Generation

The AES encrypted configuration is present in the .data section with size 0x1c0, the key to decrypt the configuration is "A5 C6 63 63 84 F8 7C 7C 99 EE 77 77 8D F6 7B 7B 0D FF F2 F2 BD D6 6B 6B B1 DE 6F 6F 54 91 C5 C5".

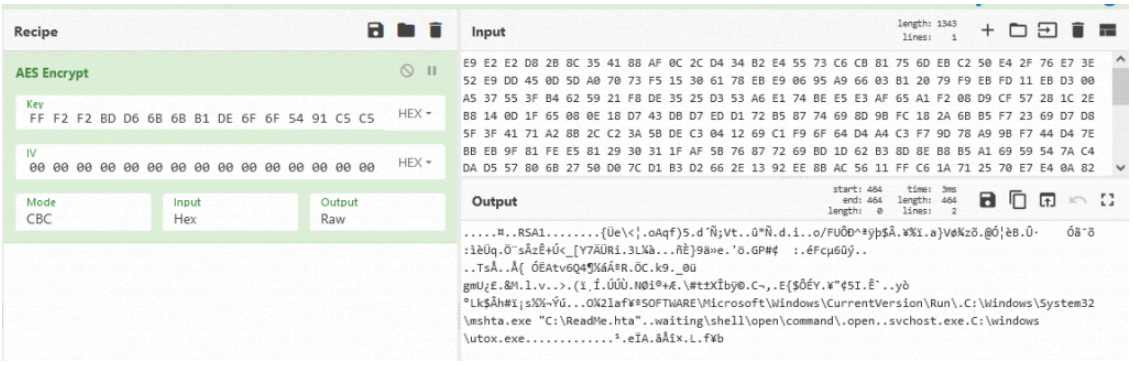

Figure 2: Stage – 1 configuration

The extracted configuration contains the RSA public key, persistence registry, key and data value for ransom note, native process to inject the next stage into and location to drop the uTox client respectively.

Once the configuration is decrypted, it checks the value "*en*" under subkey *".waiting\\shell\\open\\command\\"*, if found, the corresponding data is the RSA public key and value "n" contains RSA private key encrypted using the public key present in the configuration.

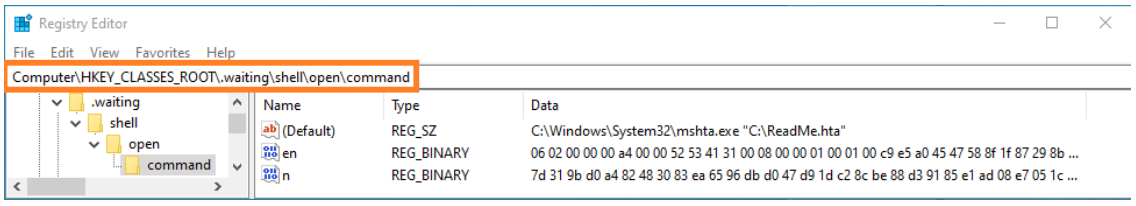

Figure 3: RSA Key Pair saved in registry

If registry value is not found, a key pair is generated using CryptGenKey API with Algid (0x1 – RSA key exchange). The private key is exported to memory using CryptExportKey with dwBlobType parameter set as 0x7(PRIVATEKEYBLOB) and it is encrypted in chunks of 0xF4 bytes using CryptEncrypt. The public key is exported in a similar manner using CryptExportKey with dwBlobType parameter set 0x6(PUBLICKEYBLOB).

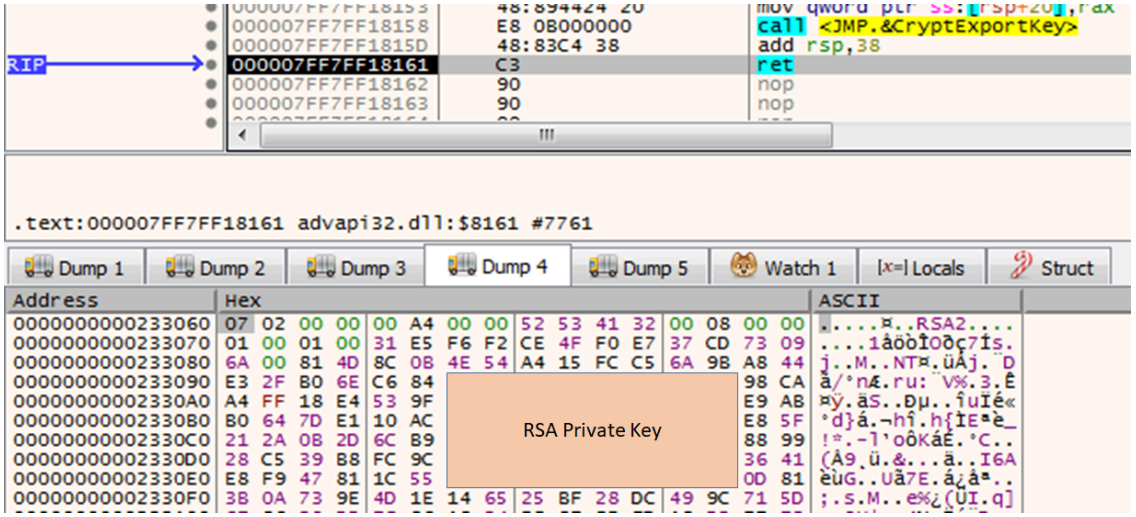

Figure 4: Private key stream

Process Injection

After the generation of public and private key pairs, the malware enumerates all the active processes and targets the first svchost.exe process to inject into. The shellcode is injected into this target process, using the conventional API's VirtualAllocEx, WriteProcessMemory, and NtCreateThreadEx is invoked to execute the shellcode in a new thread.

Stage – 2

Deleting the Trace

The injected shellcode checks if the target process has "SeDebugPrivilege" Enabled. If it is, then the Access Token is updated to NTAuthority/SYSTEM. It waits until the stage-1 process exits, to obtain a handle to the stage – 1 file. It reads the stage – 1 file from disk using MapViewOfFile, copies 0x4400 bytes from offset 0x135CA4 into a

new heap which is nothing but the stage – 2's encrypted configuration. Probably to evade memory forensic, the stage-1 file is completely filled with NULL bytes and saved, before deleting it from disk.

Stage – 2 Configuration

The stage-2 configuration is decrypted using AES with key "*50 60 30 30 03 02 01 01 A9 CE 67 67 7D 56 2B 2B 19 E7 FE FE 62 B5 D7 D7 E6 4D AB AB 9A EC 76 76*". The extracted configuration contains a bat file and its name to be dropped on disk and executed.

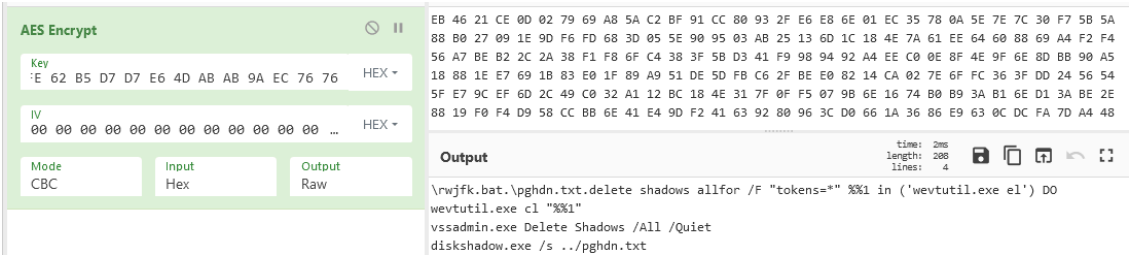

Figure 5: Stage – 2 Configuration

Deleting Shadow Copies

The bat file to delete the volume shadow copies is dropped in the Windows directory and executed using ShellExecute.

Process Injection into Explorer and Svchost

The shellcode enumerates all the active processes and on each enumeration ROR13 hashes the process name. If the calculated hash is equal to 0xDCF164CD(*EXPLORER.EXE*) or 0x561F1820(SVCHOST.EXE), but for svchost, it performs the following to target only specific service.

1. Obtain the handle using OpenProcess
2. Retrieves the PEB of the target process using NtQueryInformationProcess with ProcessInformationClass parameter set to 0.
3. Reads the cmdline argument from target process PEB using ReadProcessMemory

If the cmdline argument of the target process contains the parameter "*netsvcs*", it is chosen for the injection of final stage shellcode. The Process id of the identified target process is copied to a Heap, followed by the encrypted final stage payload which is present in the stage-1 resource section under RCDATA.

A mutex with name "itkd< 4_characters_generated_based_on_targetPID>" is created, then the encrypted resource data is decrypted using the same AES Key "*50 60 30 30 03 02 01 01 A9 CE 67 67 7D 56 2B 2B 19 E7 FE FE 62 B5 D7 D7 E6 4D AB AB 9A EC 76 76*" used before. The decrypted payload is the final stage shellcode which is injected into target process and executed using NtCreateThreadEx

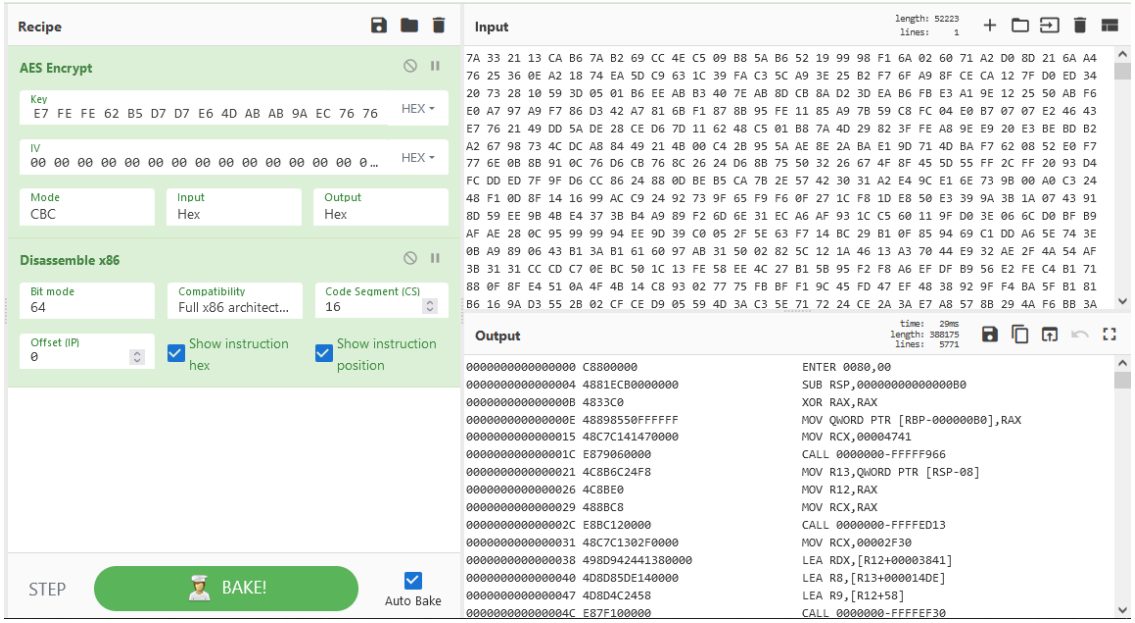

Figure 6: Final Stage Shellcode

Stage – 3 (Final Ransomware)

The Final stage creates a new heap, and decrypts another configuration which is present at offset 0x14FF with size 0x2F11 using the same AES key used in stage – 1. The configuration contains the entire ransom note which is dropped to disk in .hta format, the same public key present in stage-1 configuration and the extension to encrypt files with.

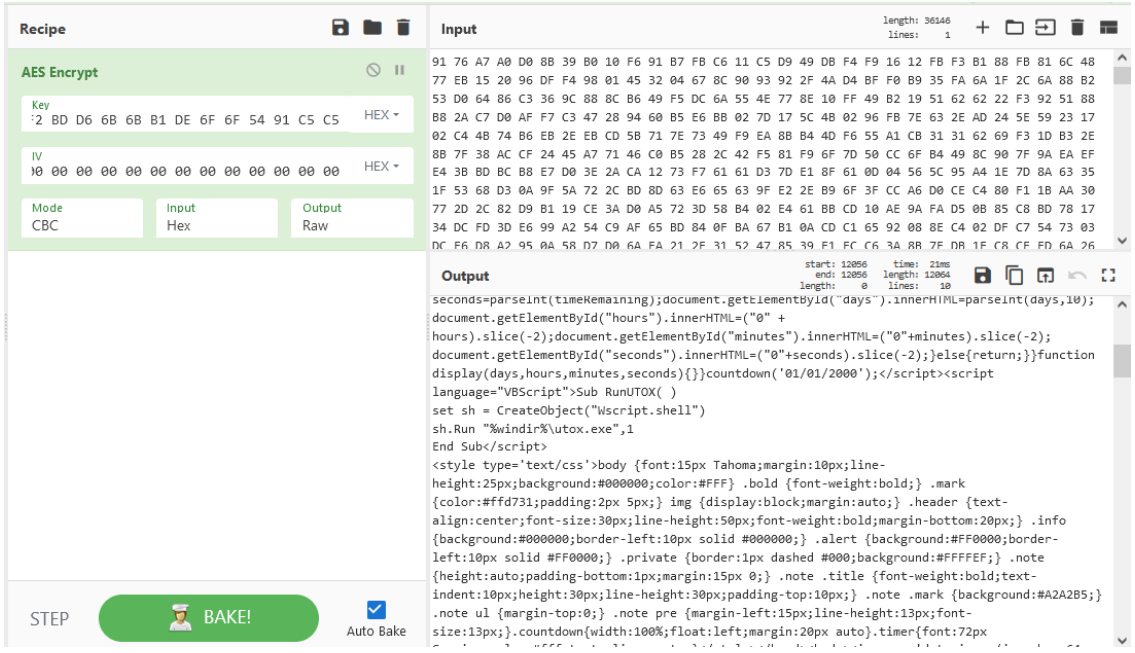

Figure 7: Ransomware Configuration

A new thread is created for each logical disk, the files are encrypted using AES algorithm, with a new private key generated for every file and it is encrypted with the hardcoded public key and appended at the end of each file. The files are encrypted with the .waiting extension. The uTox application allows the victims to communicate with the attacker with the unique id displayed in the ransom note.

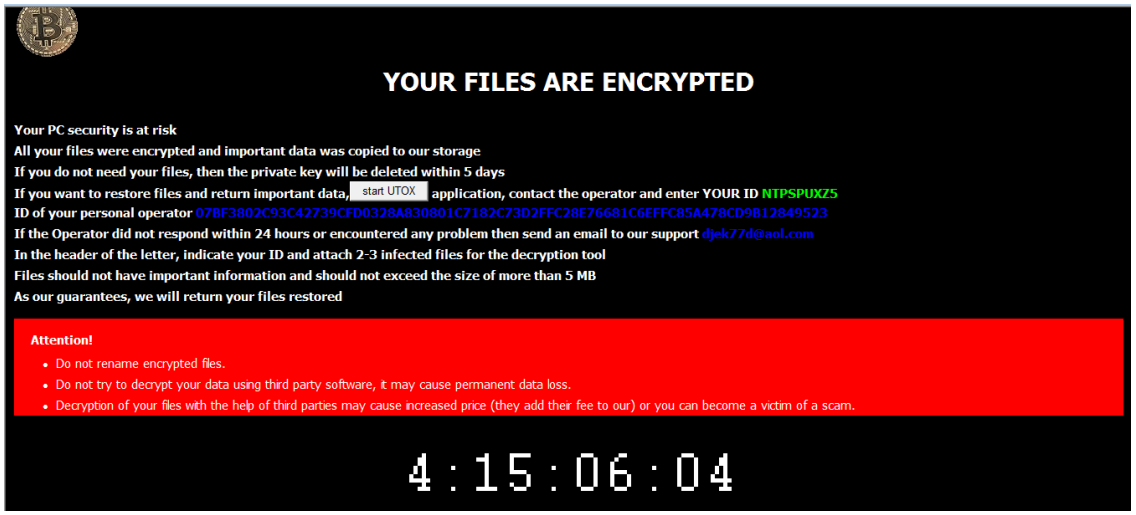

Figure 8: Crytox Ransom Note

IOCs

Hash: 823E4C4E47E8DABE32FC700409A78537

K7 Detection Name: Trojan (00564c011)

References

1. https://www.zscaler.com/blogs/security-research/technical-analysis-crytox-ransomware

Appendix A (Dynamically Resolved API's)

AdjustTokenPrivileges	0x34F2E741		RtlMoveMemory	0x97465417
CryptAcquireContextA	0x3F954B63		Sleep	0x32661A6D
CryptDestroyKey	0xD7397F82		TerminateProcess	0xB92BD08
CryptEncrypt	0x835A425D		UnmapViewOfFile	0x672A2B80
CryptExportKey	0x16E52981		VirtualAllocEx	0xD18887FC
CryptGenKey	0x8483E097		VirtualFreeEx	0x4F2BA5CE
CryptImportKey	0xC052981		VirtualProtectEx	0x94955ED7
LookupPrivilegeValueA	0x43AA560B		WaitForSingleObject	0x2671BB8F
OpenProcessToken	0xA3628BFF		WriteFile	0x70E3C54A
RegCloseKey	0x56F03636		WriteProcessMemory	0xF6E87FBA
RegCreateKeyA	0x5E723FC0		lstrcatA	0x8A1D9BCA

RegOpenKeyExA	0xFDE81F1E		lstrcmpiA	0xB1DC3443
RegQueryValueExA	0x7829A4A1		lstrcmpiW	0x61DC3443
RegSetValueExA	0x170C3FCB		NtCreateThreadEx	0x58A71ECB
CloseHandle	0xF2B7C89A		NtQueryInformationProcess	0xE650C32F
CreateFileA	0x9EB8EB8F		CreateFileW	0x4EB8EB8F
CreateFileMappingA	0x87C4720C		FileTimeToSystemTime	0x74C1905A
CreateMutexA	0xD648D4DD		FindClose	0x92A140B
CreateRemoteThread	0x4583365E		FindFirstFileW	0xD7CE34E1
CreateThread	0xE888AE7A		FindNextFileW	0xD1FDC87F
CreateToolhelp32Snapshot	0x99F5245		GetDateFormatA	0x82D70B24
DeleteFileA	0xA2EDAD8F		GetLogicalDrives	0xBA21023
GetExitCodeThread	0xFBD76D17		GetSystemTimeAsFileTime	0x8FBB53E7
GetFileSize	0x4966632A		GetSystemTimes	0xFE2CDA22
GetLastError	0x87E43BC		GetTickCount	0x20841296
GetWindowsDirectoryA	0x63061FFC		GlobalMemoryStatus	0x74C9FD10
GlobalAlloc	0x5287A129		MoveFileW	0x9BDBE590
GlobalFree	0x8CEF887D		ReadFile	0xCE2BC47E
LoadLibraryA	0x2EB89E41		SetEndOfFile	0xCC719466
MapViewOfFile	0xA48A2B6F		SetFileAttributesW	0xB0DB724A
OpenProcess	0xBF2A3840		SetFilePointerEx	0xD90CDB68
Process32First	0x6CB1F1E6		SetThreadPriority	0x704F3375
Process32Next	0x2D65D010		ShellExecute	0x3A6952BF
ReadProcessMemory	0xF08369FA		lstrcatW	0x3A1D9BCB
ReleaseMutex	0x36C87830		lstrlenW	0x38A62BCB

Source: https://labs.k7computing.com/index.php/encrypted-chaos-analysis-of-crytox-ransomware/