The Road to Ransomware Resilience, Part 2: Behavior Analysis

By CyCraft Technology Corp

Published: 2022-06-10 · Archived: 2026-04-05 14:50:32 UTC

Press enter or click to view image in full size

Understanding Active and Emerging Threats & Developing a More Effective Novel Response

*This is the second in a series of articles. Follow this link to read from the beginning.

The road to constructing an effective novel response against ransomware begins with understanding the recent trends of the underground ransomware ecosystem: common targets of ransomware, typical ransomware behavior, common ransomware encryption schemes, as well as the construction and application of decryptors and other effective response approaches and tools.

In this upcoming series of articles, we will discuss each of these factors one by one.

Ransomware Behavior Trends

We've looked at trends in victim selection as well as the underground ecosystem; however, what has proven most interesting has been the trends within the ransomware itself.

We analyzed 9 different ransomware and focused on these aspects of ransomware behavior: triggers, evasion and obfuscation techniques, and encryption schemes.

Press enter or click to view image in full size

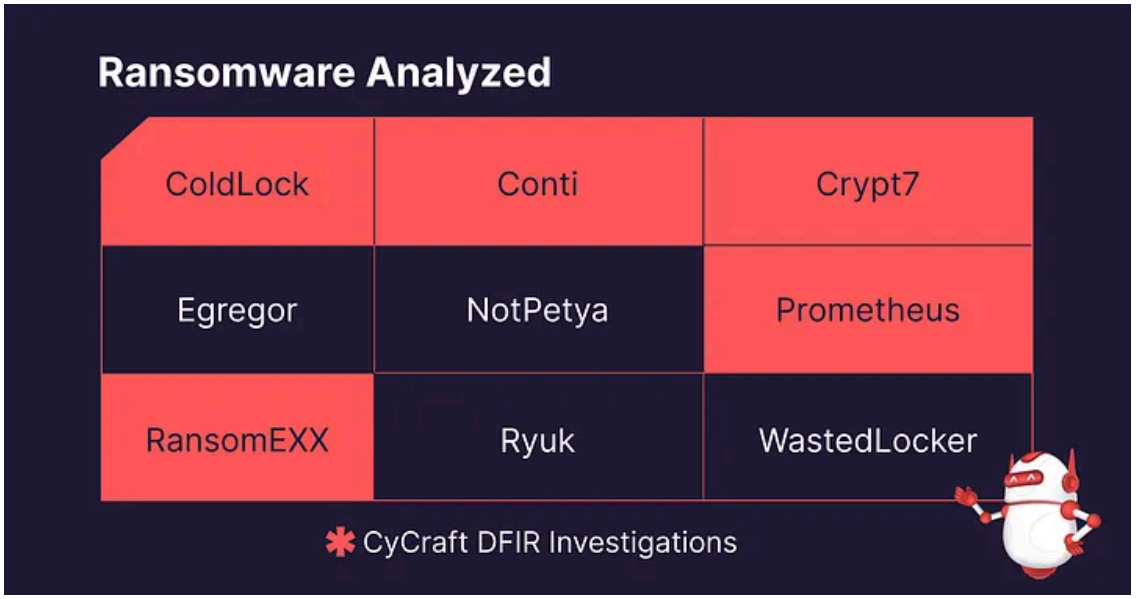


Fig. 2 — Ransomware Analyzed

Trigger Analysis

Ransomware typically wants to encrypt and extract as many files as it can, as fast as it can, as long as it can; however, it needs to first ensure the targeted files are their worth encrypting — whatever the attackers' motivations might be. In order to accomplish all of this, prior to execution, ransomware will typically perform environment and atomic execution checks.

Environment Check

Ransomware will check the environment it's currently located in to ensure it is not located in a sandbox, honeypot, or other virtual machine but in the target victim's real environment. "Big game" targets typically have more mature defenses, meaning that ransomware only has one chance to trigger execution.

Press enter or click to view image in full size

Fig. 3 — WastedLocker Environment Check

Wastedlocker, for example, checks for a specific interface to ensure the environment it's currently located in is real and not a virtual machine.

Atomic Execution Check

Ransomware developers are constantly looking for ways to increase encryption efficiency — encrypt more files in less time. One method commonly used across different ransomware is an atomic execution check that ensures files selected for encryption are only encrypted once.

Some take a more streamlined approach with a static mutex, such as ColdLock, Crypt7, and Prometheus. On the other hand, some ransomware took a more complicated approach with a dynamic mutex. RansomEXX generated the mutex name via MD5 hashing the endpoint's computer name while Prometheus went a step further and directly hardcoded the process name into the code, and then later double-checked the targeted process name to ensure targeted files wouldn't be double encrypted.

Press enter or click to view image in full size

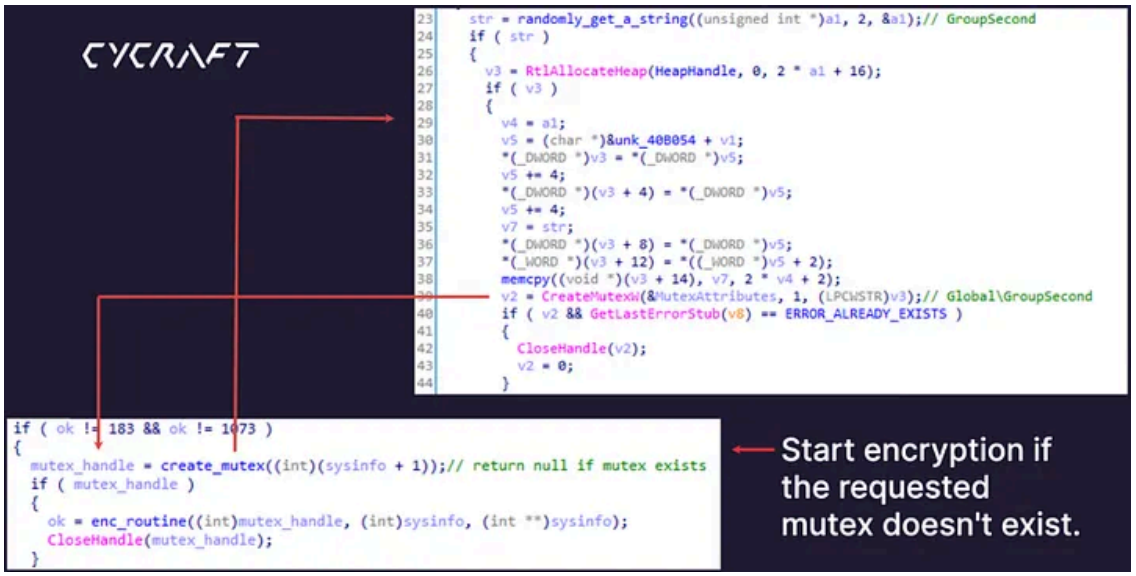


Fig. 4 — WastedLocker Atomic Execution Check

WastedLocker (pictured above) also took a much more dynamic approach with its mutex by only beginning encryption if the newly created mutex were “unique” for each endpoint. While WastedLocker seemed to “randomly” generate “unique” mutex names; however, this “random” mutex name was actually deterministic on the same endpoint and could therefore be reversed. That is, different endpoints would have different mutex names (“random” across “different endpoints”, making it harder for EDR/AV to detect), but the same endpoint would always have the same mutex name (so it wouldn’t double encrypt on the same endpoint).

For your reference, we created the chart below to help you better compare each of the studied ransomware samples.

Press enter or click to view image in full size

Attack Date	Ransomware	Environment Check	Atomic Execution Check
2017 Jun	NotPetya	N/A	Checked for the existence of file "C:\Windows\perfc"
2018 Aug	Ryuk	N/A	N/A
2020 May	ColdLock	Time bomb - encrypted at 12:10 pm (lunch break).	Static Mutex - "8c861dd7-ef9b-41c6-b59c-c7cd94434d01"
2020 Jul	WastedLocker	Check registry for specific interface - interface\{b196b287-bab4-101a-b69c-00aa00341d07}	Dynamic Mutex - Global$STR where string $STR chosen from registry HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control
2020 Oct	Egregor	Check for languages native to CIS countries	N/A
2020 Nov	Conti	N/A	Static Mutex - "kjkbmusop9iqkamvcrewuyy777"
2021 Mar	Crypt7	N/A	1. Static Mutex - "W@nn@Cry_1.0" 2. Registry Value - HKEY_LOCAL_MACHINE\Software\ODBC\State
2021 Apr	Prometheus	1. Check Low DiskSize 2. Check for VM 3. Check for SbieDll.dll 4. Check Remote Debugger	1. Static Mutex - "3b257655-f219-48a9-a4db-c57417cd780b 2. Check for duplicated process name
2021 Jul	RansomEXX	Check for files accessed within the last 2 years	Dynamic Mutex - StringFrom GUID2(MD5($computer_name))

Fig. 5 — Ransomware Trigger Behavior

Idiosyncratic Checks

While different strains of ransomware share similarities due to the specific nature of ransomware attacks, ransomware is still a reflection of the more unique motivations driving the developers.

Egregor, for example, checks for languages native to the Ex-Soviet CIS countries (Commonwealth of Independent States); this is most likely due to Egregor developers being located in, or from, one of the CIS countries and wanting to avoid attention from local law enforcement agencies.

When asked if they check the location of their victim, the leader of the LockBit ransomware group responded, “Yes, that’s a code of honor. You cannot attack your own nation. I was born in the Soviet Union.”[8]

Some checks could have been developed and implemented due to the cultural environment of the target as opposed to their digital environment. In the ColdLock case[5], the attackers waited for a long holiday weekend to allow the GPO to automate the distribution of the ColdLock ransomware throughout the entire system, maximizing the number of affected endpoints for the day of the attack. When the ransomware did trigger, it was at precisely 12:10 in the afternoon — Monday lunch break.

Compared to the rest of the analyzed ransomware, Prometheus was far more meticulous and sophisticated in both its environment and atomic checks.

However, some ransomware took a much more complicated approach; WastedLocker and RansomEXX both leveraged a dynamic mutex, with WastedLocker being the most dynamic and complex.

Evasion and Obfuscation Techniques

Ransomware follows the design philosophy of encrypting and extracting as many files as it can, as long as it can. While environment and atomic checks help ensure targeted files get encrypted as efficiently as possible, obfuscation and evasion techniques help ensure the attackers have enough time to locate and encrypt sensitive files.

However, modern ransomware developers think in the long term. The longer it takes defense teams to reverse engineer ransomware and respond, the longer the ransomware gang can stay operational.

Here are some highlighted obfuscation techniques.

Conti Ransomware: API Unhooking

API Hooking is a common technique used by both AV and EDR solutions in order to monitor processes and code behavior in near real-time and is typically done by adding jump instructions in front of functions the system wants to monitor.

MITRE ATT&CK (T1056.004) API HOOKING
Adversaries may hook into Windows application programming interface (API) functions to collect user credentials. Malicious hooking mechanisms may capture API calls that include parameters that reveal user authentication credentials. Unlike Keylogging, this technique focuses specifically on API functions that include parameters that reveal user credentials. Hooking involves redirecting calls to these functions and can be implemented via hooks procedures, import address table (IAT) hooking, or inline hooking.

Conti (pictured below) bypassed this by meticulously searching for jump opcode. If found, Conti would attempt to patch the jump opcode and unhook the API, leaving the AV or EDR solutions blind to Conti’s behavior.

Press enter or click to view image in full size

```
if ( !(unsigned int)init_import_cache() )
  FreeLibraryAndExitThread(hModule, 1u);
load_libraries_and_check_inline_hook();
```

```
func_body_ptr = (LPCBYTE)GetProcAddress(hModule, export_func_name);
if ( func_body_ptr )
{
  v23 = *func_body_ptr;
  if ( v23 == 0xE9u || v23 == 0xFFu && func_body_ptr[1] == 0x25 )// check jmp opcode
  {
    if ( v18 )
    {
      v24 = (BYTE *)v18;
      while ( 1 )
      {
        v25 = func_body_ptr[(_QWORD)v24 - v18];
        if ( *v24 < v25 || *v24 > v25 )
          break;
        if ( (unsigned __int64)&(++v24)[-v18] >= 2 )
          goto LABEL_35;
      }
    }
    flOldProtect = 0;
    v31 = 0;
    if ( !VirtualProtect((LPVOID)func_body_ptr, 0x40ui64, 0x40u, &flOldProtect) )
      return;
    *(_QWORD *)func_body_ptr = *(_QWORD *)v18;// unhook function by copying 16bytes
    *((_WORD *)func_body_ptr + 4) = *(_WORD *)(v18 + 8);
    v10 = (IMAGE_EXPORT_DIRECTORY *)flOldProtect;
    if ( !VirtualProtect((LPVOID)func_body_ptr, 0x40ui64, flOldProtect, &v31) )
      return;
  }
```

Fig 6 — Conti Ransomware API Unhooking

Note the two highlighted green sections labeled “check jmp code” and “unhook function by copying 16bytes”. The first part of the code checks whether the opcode is a “jump” instruction, which is a potential sign of API hooking. Then, the second part of the code overwrites these instructions into ‘’nop” instructions to remove the hook.

Conti Ransomware: String Obfuscation

Conti ransomware leveraged more than 100 string obfuscation functions in our sample alone. The particular obfuscation method pictured below is quite simple to decrypt manually; however, this quickly becomes burdensome when tasked to manually decrypt over 100 string obfuscation functions. In this particular case, a script was developed in order to automate the decryption process.

Press enter or click to view image in full size

```
v31 = 0;
v32 = 3988339979279296123i64;
v33 = 97;
v34 = 97;
v35 = 38;
a1 = LoadLibraryA(&v49);
v2 = 0i64;
do
{
  *((_BYTE *)&v32 + v2) = (24 * (38 - *((unsigned __int8 *)&v32 + v2)) % 127 + 127) % 127;
  ++v2;
}
while ( v2 < 0xB );
v3 = LoadLibraryA(&v32);
```

```
v48 = 0;
v0 = 0i64;
v49 = 20;
v50 = 9;
v51 = 54;
v52 = 89;
v53 = 9;
v54 = 43;
v55 = 2;
v56 = 106;
v57 = 14;
v58 = 113;
v59 = 43;
v60 = 43;
v61 = 99;
v1 = 0i64;
do
{
  *(&v49 + v1) = (11 * (99 - (unsigned __int8)*(&v49 + v1)) % 127 + 127) % 127;
  ++v1;
}
while ( v1 < 0xD );
v31 = 0;
v32 = 3988339979279296123i64;
v33 = 97;
v34 = 97;
v35 = 38;
a1 = LoadLibraryA(&v49);
```

Fig. 7 — Conti Ransomware String Obfuscation

Prometheus Ransomware: GetString

Unlike Conti, Prometheus is implemented in managed code (.NET MSIL) and did not develop their own obfuscation method but rather used the commercial obfuscator SmartAssembly v7.5.2.4508. While this version of SmartAssembly gives Conti access to numerous obfuscation techniques, we wanted to highlight Prometheus' use of the GetString function.

Get CyCraft Technology Corp's stories in your inbox

Every string reference has been replaced by the GetString function. From the code below, we see that function .\u001F is, in fact, the GetString function.

Press enter or click to view image in full size

```
string text = managementBaseObject[QNNiaBjLhIIsGXT.\u001F(107380643)].ToString().ToLower();
if ((text == QNNiaBjLhIIsGXT.\u001F(107380082) && managementBaseObject[QNNiaBjLhIIsGXT.\u001F
  (107380053)].ToString().ToUpperInvariant().Contains(QNNiaBjLhIIsGXT.\u001F(107380044))) ||
  text.Contains(QNNiaBjLhIIsGXT.\u001F(107380063)) || managementBaseObject[QNNiaBjLhIIsGXT.
  \u001F(107380053)].ToString() == QNNiaBjLhIIsGXT.\u001F(107380022))
{
    return true;
}
```

Fig. 8 — Prometheus Ransomware GetString Obfuscation

```
// Token: 0x0400008A RID: 138
[NonSerialized]
internal static GetString \u001F;
```

Fig. 9 — Prometheus Ransomware GetString Function Revealed

In addition, every class within Prometheus ransomware has its own GetString function.

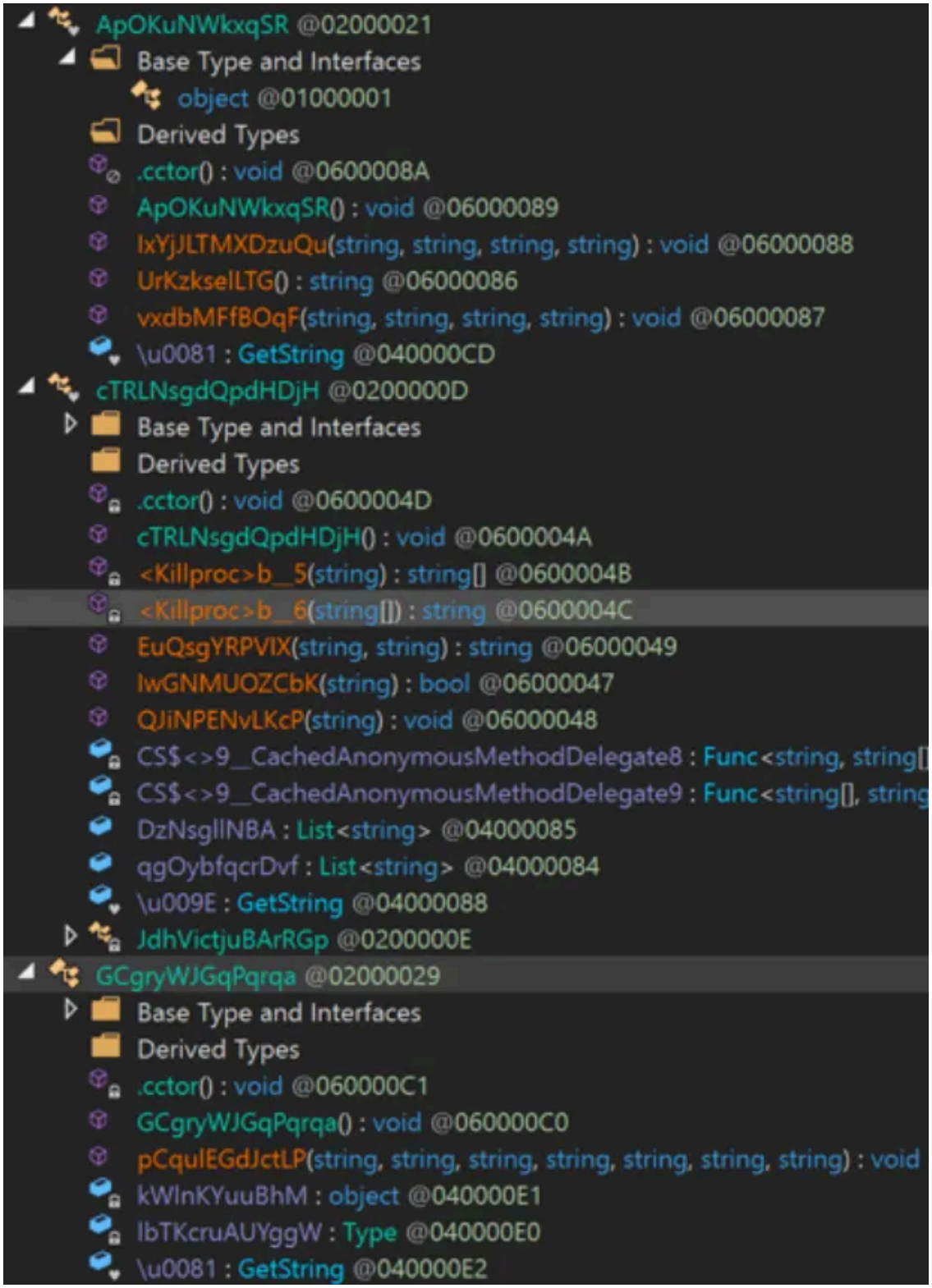


Fig. 10 — Prometheus Ransomware with Multiple GetString Functions

Press enter or click to view image in full size

```
// SmartAssembly.StringsEncoding.Strings
// Token: 0x0600010E RID: 270 RVA: 0x0001F070 File Offset: 0x0001D270
public static string Get(int A_0)
{
    A_0 ^= 107396847;
    A_0 -= Strings.offset;
    if (!Strings.cacheStrings)
    {
        return Strings.GetFromResource(A_0);
    }
    return Strings.GetCachedOrResource(A_0);
}
```

Fig. 11 — Prometheus Ransomware GetString Function Simplicity

As seen directly above, the GetString function itself is not complicated — just a few integer operations. After the ransomware has offset the string, it will reference resource stream {c4633a62–8069–4a7e-9e5d-1429bccb887a}, which after unzipped and decoded from the revealed Base64 format, will reveal the final string buffer where every string used by the ransomware is located.

Press enter or click to view image in full size

```
// SmartAssembly.HouseOfCards.Strings
// Token: 0x06000103 RID: 259 RVA: 0x0001EF08 File Offset: 0x0001D108
[\u0001]
public static void CreateGetStringDelegate(Type ownerType)
{
    foreach (FieldInfo fieldInfo in ownerType.GetFields(BindingFlags.Static | BindingFlags.NonPublic |
      BindingFlags.GetField))
    {
        try
        {
            if (fieldInfo.FieldType == typeof(GetString))
            {
                DynamicMethod dynamicMethod = new DynamicMethod(string.Empty, typeof(string), new Type[]
                {
                    typeof(int)
                }, ownerType.Module, true);
                ILGenerator ilgenerator = dynamicMethod.GetILGenerator();
                ilgenerator.Emit(OpCodes.Ldarg_0);
                foreach (MethodInfo methodInfo in typeof(Strings).GetMethods(BindingFlags.Static |
                  BindingFlags.Public))
                {
                    if (methodInfo.ReturnType == typeof(string))
                    {
                        ilgenerator.Emit(OpCodes.Ldc_I4, fieldInfo.MetadataToken & 16777215);
                        ilgenerator.Emit(OpCodes.Sub);
                        ilgenerator.Emit(OpCodes.Call, methodInfo);
                        break;
                    }
                }
                ilgenerator.Emit(OpCodes.Ret);
                fieldInfo.SetValue(nul                                    eof(GetString)));
```

```
index1 = (((107380063 - (0x400008A & 0xffffff) ) ^ 107396847) - 26 ) = 0xc220
index: C220, size: 8, str: vmware
index1 = (((107380643 - (0x400008A & 0xffffff) ) ^ 107396847) - 26 ) = 0xc1dc
index: C1DC, size: 10, str: Manufacturer
```

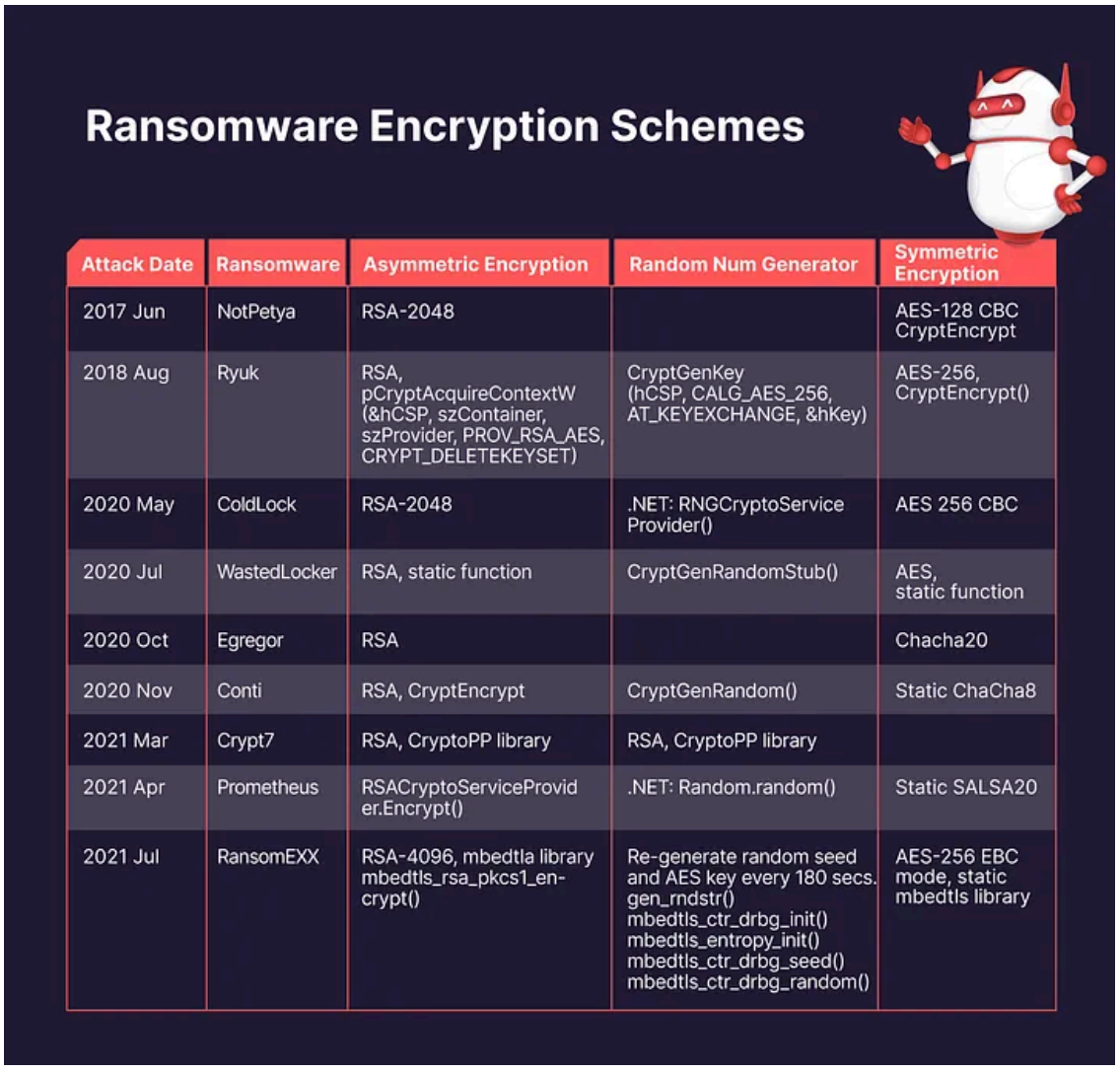
Fig. 12 — Prometheus Ransomware CreateGetStringDelegate Function

The CreateGetStringDelegate function generates IL code during runtime and dynamically adjusts arguments pasted into GetString. This approach allows for not only every GetString function to have a unique metadata token but for each class to also have a unique formula to calculate the string index.

Ransomware Encryption Schemes

Ransomware we analyzed typically followed the same two-layered encryption scheme seen in most ransomware. The first layer is file encryption leveraging symmetric encryption algorithms (such as AES); typically, a unique file encryption key is generated for each encrypted file. The second layer is the key encryption, often leveraging asymmetric encryption algorithms (such as RSA) and encrypting the file encryption key. The public key is used to encrypt the file encryption key on the victim's computer while the private key is kept by the attackers.

Press enter or click to view image in full size

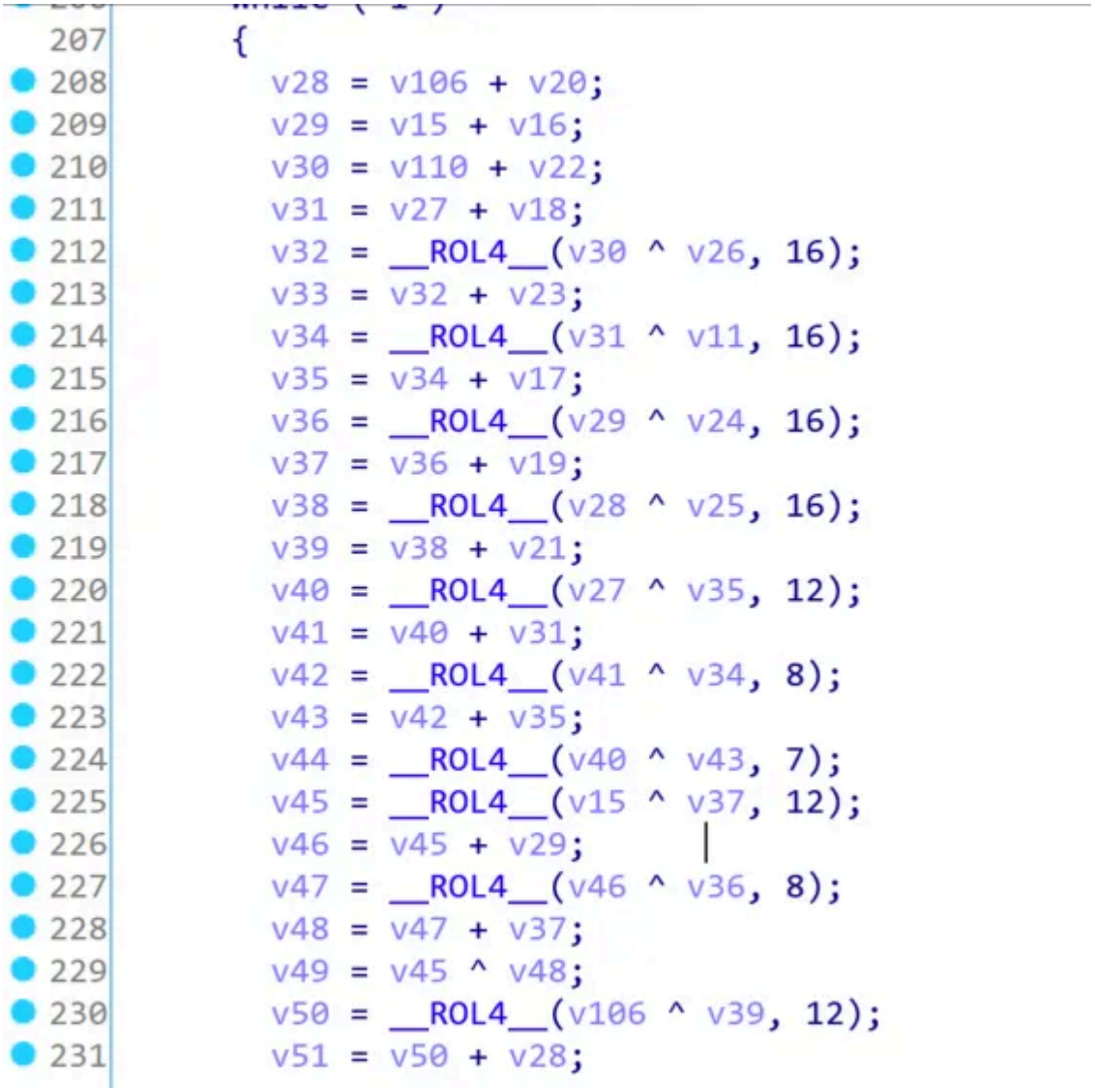


Attack Date	Ransomware	Asymmetric Encryption	Random Num Generator	Symmetric Encryption
2017 Jun	NotPetya	RSA-2048		AES-128 CBC CryptEncrypt
2018 Aug	Ryuk	RSA, pCryptAcquireContextW (&hCSP, szContainer, szProvider, PROV_RSA_AES, CRYPT_DELETEKEYSET)	CryptGenKey (hCSP, CALG_AES_256, AT_KEYEXCHANGE, &hKey)	AES-256, CryptEncrypt()
2020 May	ColdLock	RSA-2048	.NET: RNGCryptoService Provider()	AES 256 CBC
2020 Jul	WastedLocker	RSA, static function	CryptGenRandomStub()	AES, static function
2020 Oct	Egregor	RSA		Chacha20
2020 Nov	Conti	RSA, CryptEncrypt	CryptGenRandom()	Static ChaCha8
2021 Mar	Crypt7	RSA, CryptoPP library	RSA, CryptoPP library	
2021 Apr	Prometheus	RSACryptoServiceProvider.Encrypt()	.NET: Random.random()	Static SALSA20
2021 Jul	RansomEXX	RSA-4096, mbedtla library mbedtls_rsa_pkcs1_encrypt()	Re-generate random seed and AES key every 180 secs. gen_rndstr() mbedtls_ctr_drbg_init() mbedtls_entropy_init() mbedtls_ctr_drbg_seed() mbedtls_ctr_drbg_random()	AES-256 EBC mode, static mbedtls library

Fig. 13 — Ransomware Encryption Schemes

Here, you can see that RSA is the most commonly used asymmetric encryption and AES the most common choice for symmetric encryption.

Some ransomware (as pictured above) used system built-in APIs, such as CryptEncrypt and Random.random(). Ryuk's use of AES-256 and CryptEncrypt() for symmetric encryption and Conti's use of CryptGenRandom() for RNG (random number generation) are good examples of this. Some ransomware, such as Conti, used statically linked libraries for cryptography operations.

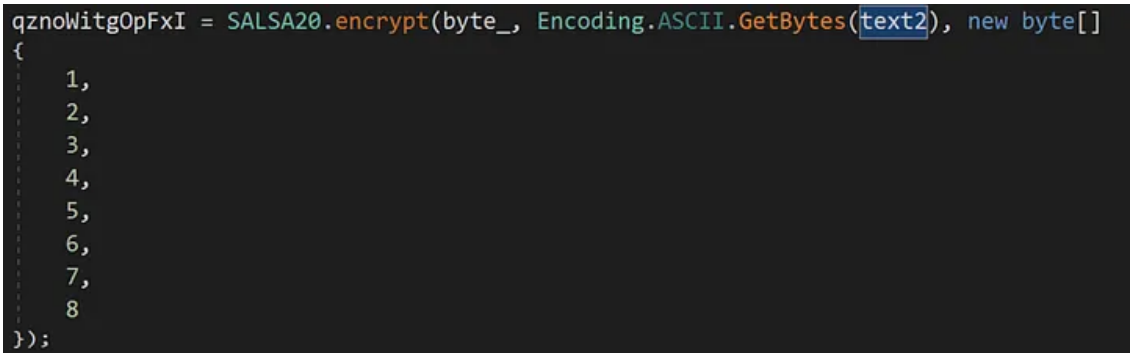


```
207     {
208       v28 = v106 + v20;
209       v29 = v15 + v16;
210       v30 = v110 + v22;
211       v31 = v27 + v18;
212       v32 = __ROL4__(v30 ^ v26, 16);
213       v33 = v32 + v23;
214       v34 = __ROL4__(v31 ^ v11, 16);
215       v35 = v34 + v17;
216       v36 = __ROL4__(v29 ^ v24, 16);
217       v37 = v36 + v19;
218       v38 = __ROL4__(v28 ^ v25, 16);
219       v39 = v38 + v21;
220       v40 = __ROL4__(v27 ^ v35, 12);
221       v41 = v40 + v31;
222       v42 = __ROL4__(v41 ^ v34, 8);
223       v43 = v42 + v35;
224       v44 = __ROL4__(v40 ^ v43, 7);
225       v45 = __ROL4__(v15 ^ v37, 12);
226       v46 = v45 + v29;
227       v47 = __ROL4__(v46 ^ v36, 8);
228       v48 = v47 + v37;
229       v49 = v45 ^ v48;
230       v50 = __ROL4__(v106 ^ v39, 12);
231       v51 = v50 + v28;
```

Fig. 14 — Conti Ransomware's use of ChaCha8

Some ransomware leveraged encryption algorithms from other well-known malware, such as Conti's use of ChaCha8 (pictured above) or Prometheus's use of SALSA20 (pictured below). This approach was most likely taken to increase encryption efficiency.

Press enter or click to view image in full size



```
qznoWitgOpFxI = SALSA20.encrypt(byte_, Encoding.ASCII.GetBytes(text2), new byte[]
{
    1,
    2,
    3,
    4,
    5,
    6,
    7,
    8
});
```

Fig. 15 — Prometheus Ransomware's use of SALSA20

Encryption Optimization

As mentioned earlier, ransomware typically has three priorities: find sensitive data fast, leverage evasion and obfuscation techniques to prolong the attack, and encrypt targeted data quickly and efficiently.

Encryption optimization is on every ransomware developer's mind. The two main methods we have observed from our analysis were increasing the number of CPU threads and employing different encryption methods for files of different sizes.

Conti ransomware was notorious for its speed, which was due to its leveraging 32 threads. Conti isn't alone in using multiple threads. Other ransomware developers have explored this approach, such as REvil, LockBit, Rapid, Thanos, Phobos, and MagaCortex.

Other ransomware (e.g., Prometheus and REvil) increased efficiency by employing different encryption methods based on file size. Smaller files would be fully encrypted while larger files, such as image or video files, would only be partially encrypted; even encrypting only the header to make the file inaccessible could be enough to serve the attackers' purposes. Read more: Cycraft Releases Decryptor for Prometheus Ransomware

The Long and Winding Road

On our next stop on the road to ransomware resilience, we will explore defense and preventative solutions: how to gather and leverage intelligence from hacker forums, the dark web, OSINT, and other sources to identify potential enterprise-targeting threats. Learn how to prevent attacks in advance by locating leaked documents, leaked PII, credentials for sale on the dark web, enterprise-related credentials leaked from 3rd party services, and more.

In addition, we will also discuss new innovative defense approaches including AI-based threat hunting, AD environment assessments, and developing digital vaccines for ransomware.

> If you're interested in learning about compromise assessments to ensure the health and security of your network, engage with us directly at engage@cycraft.com

Everything Starts From Security

CyCraft Customers can prevent cyber intrusions from escalating into business-altering incidents. From endpoint to network, from investigation to blocking, from in-house to cloud, CyCraft AIR covers all aspects required to provide small, medium, and large organizations with the proactive, intelligent, and adaptable security solutions needed to defend from all manner of both existing and emerging security threats with real-time protection and visibility across the organization.

Engage with CyCraft

Blog | LinkedIn | Twitter | Facebook | CyCraft

Press enter or click to view image in full size

CyCraft secures government agencies, police and defense organizations, Fortune Global 500 firms, top banks and financial institutions, critical infrastructure, airlines, telecommunications, hi-tech firms, SMEs, and more by being **Fast / Accurate / Simple / Thorough.**

CyCraft powers SOCs using innovative AI-driven technology to automate information security protection with built-in advanced managed detection and response (MDR), global cyber threat intelligence (CTI), smart threat intelligence gateway (TIG) and network detection and response (NDR), security operations center (SOC) operations software, auto-generated incident response (IR) reports, enterprise-wide Health Check (Compromise Assessment, CA), and Secure From Home services. Everything Starts From Security.

Meet your cyber defense needs in the 2020s by engaging with CyCraft at engage@cycraft.com

Additional Resources

- Learn more about cybersecurity in our CyCraft Classroom Series. Our latest article, "What is Managed Detection and Response (MDR)?" teaches the benefits of MDR, its unique selling points, and how to make better-informed decisions when choosing an MDR service or vendor.
- Out of the 47 Representative AI Startups listed in Gartner's AI Market Guide, 7 are based in Taiwan, and 5 are based in Hong Kong. But only 1 of the 47 Representative AI Startups focused on cybersecurity products and services — CyCraft.
- Read our latest white paper to learn what threat actors target Taiwan, their motivations & how Taiwan organizations retain resilience against some of the most sophisticated and aggressive cyber attacks in the world.
- Is your SOC prepared for the next decade of cyber attacks? Read our latest report on building effective SOCs in the 2020s, the challenges to overcome, and the stressors to avoid — includes research from Gartner, Inc. on why Midsize enterprises are embracing MDR providers.
- New to the MITRE Engenuity ATT&CK Evaluations? START HERE for a fast, accurate, simple, thorough introductory guide to understanding the results.

- Our CyCraft AIR security platform achieved a 96.15% Signal-to-Noise Ratio with zero configuration changes and zero delayed detections straight out of the box.
