Topic:- Emotet Malware Analysis

Malware Hash:-

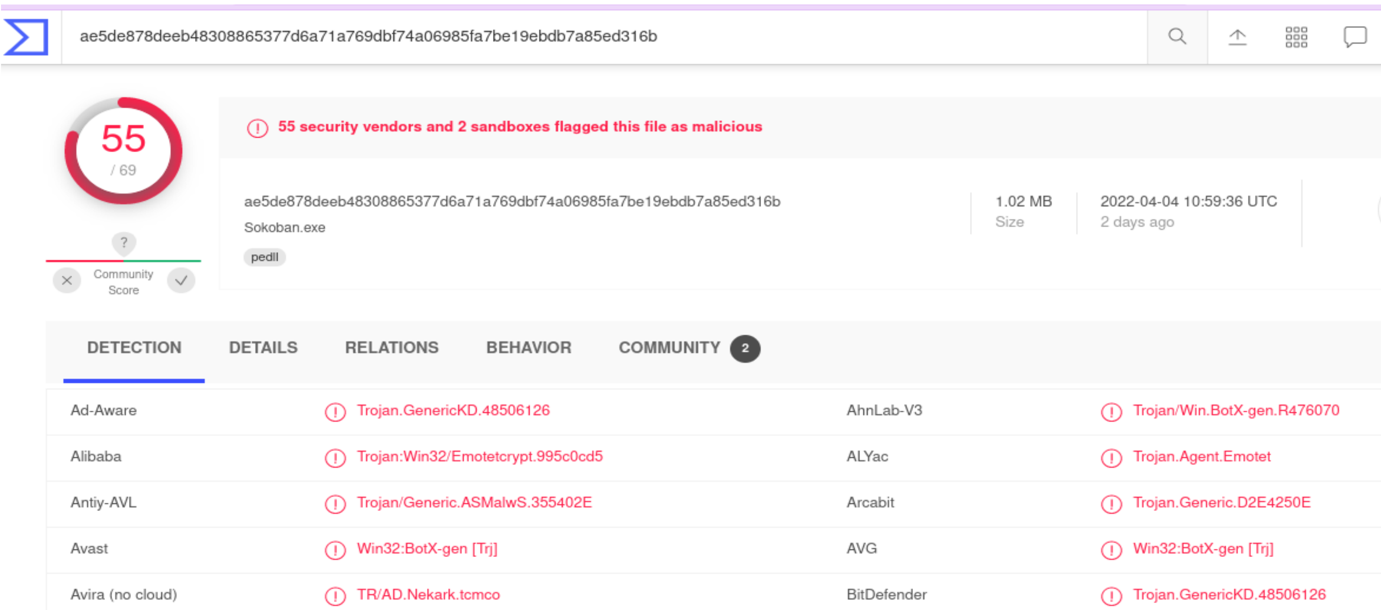
- sha256:- ae5de878deeb48308865377d6a71a769dbf74a06985fa7be19ebdb7a85ed316b

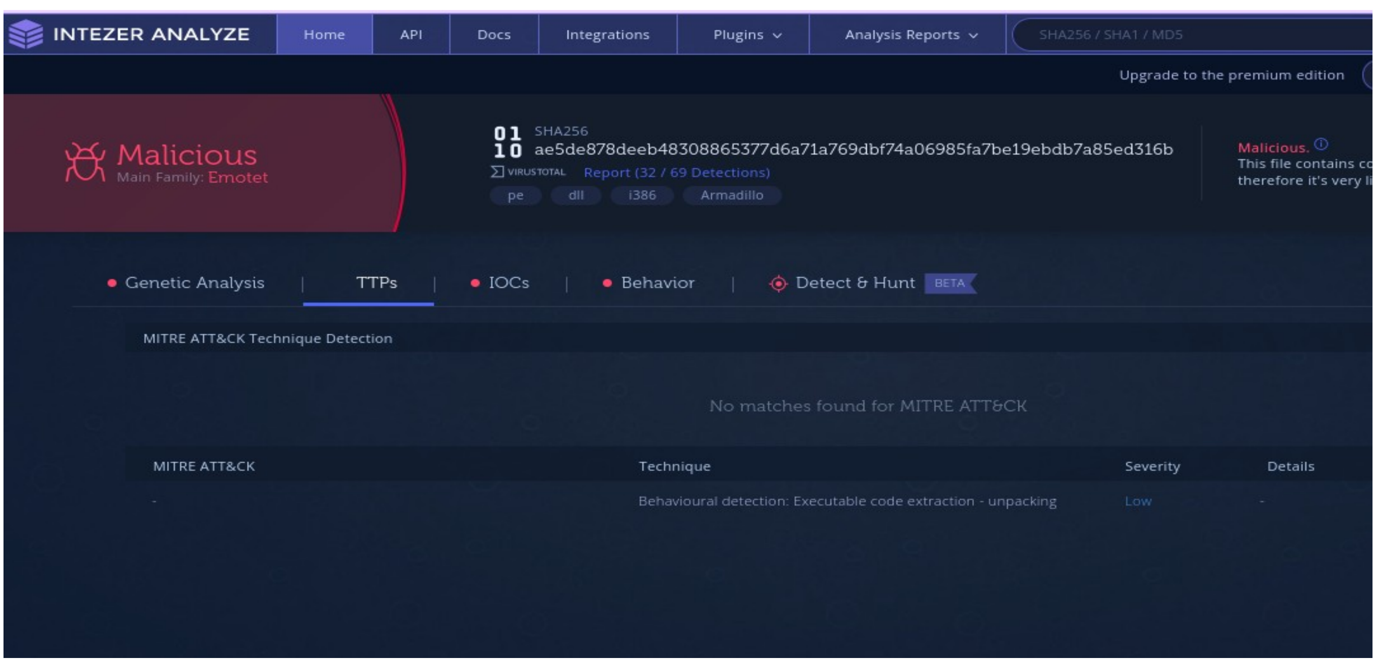
Tools:- x64dbg, Cutter, Process Hacker, Wireshark

Overview:- Emotet is a trojan primarily used in phishing attacks. It spreads mainly through email containing links or malicious attachments which act as an initial vector into the machine. While emerging in 2014 as a simple trojan with worm like capabilities, the malware has seen a resurgence in the past year and is mostly being used as as dropper or downloader of other malware. In this write-up we take a look at the various functionalities of the malware such as encryption, API address resolution and identify IOCs and IOAs.

CHECKING MALWARE SANDBOXES

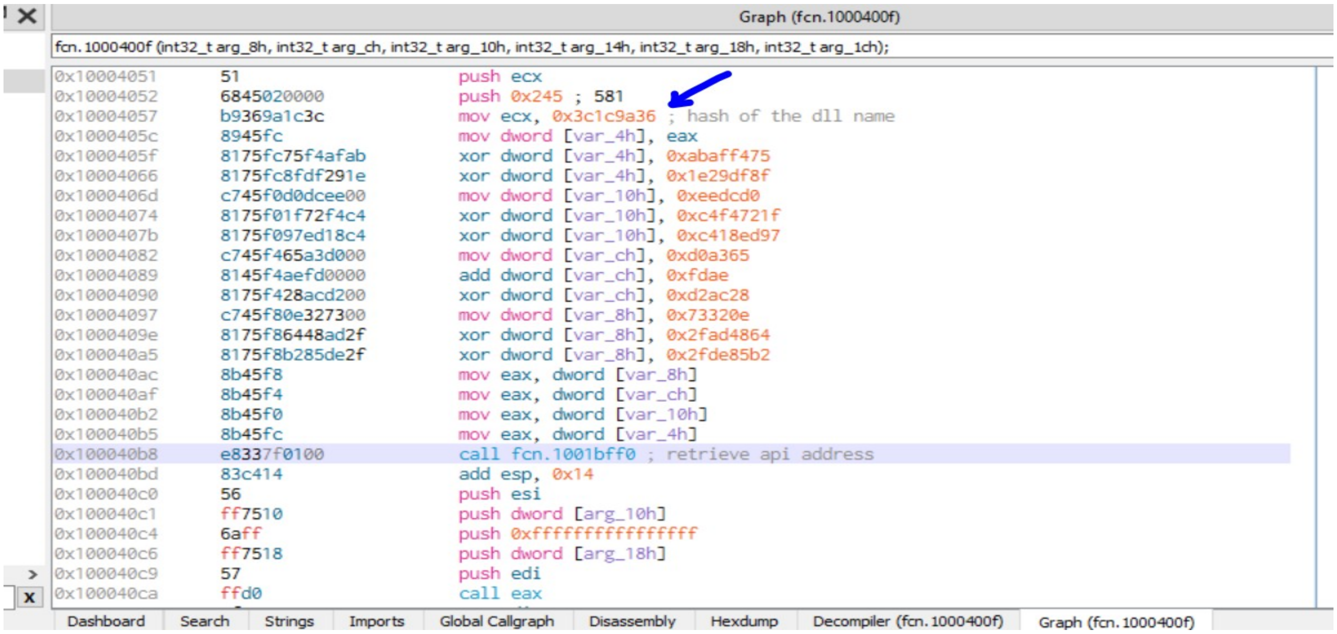
Taking a look at the Virus Total, we can see that it 3/4th of the vendors detect the the file as malicious. But if we take a look at the Intezer analysis we can see that it is not able to detect techniques or behaviour of the malware considering the malware has a lot of junk code and uses hashes for resolving API address dynamically.



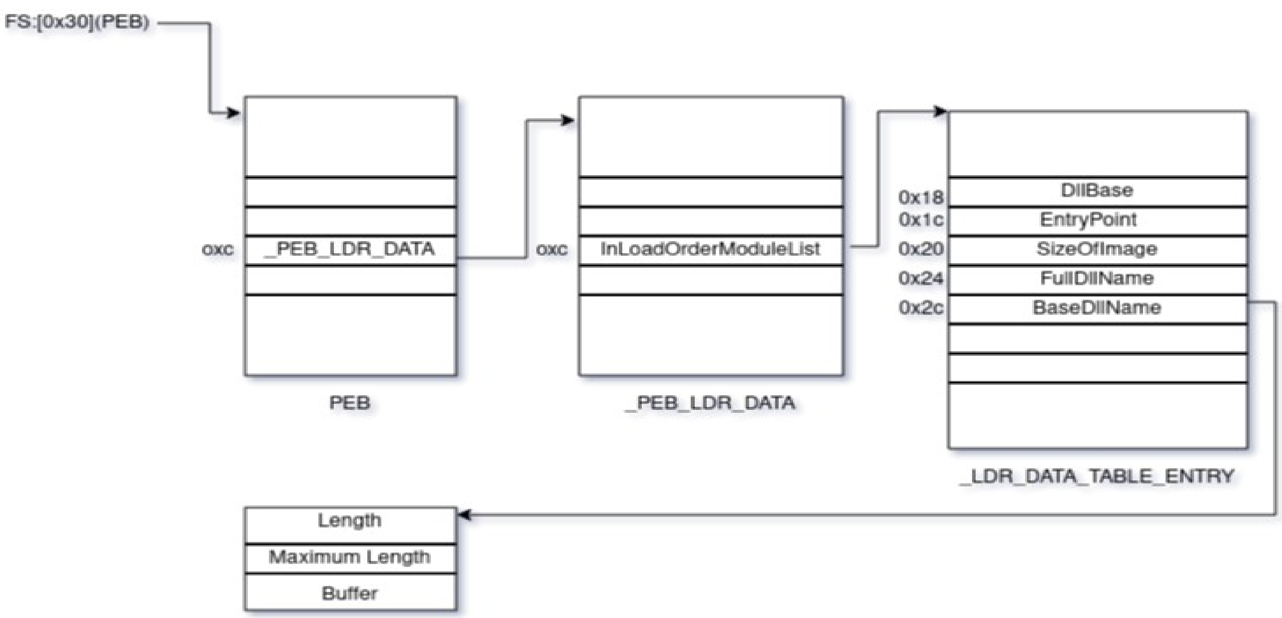


RESOLVING API ADDRESSES DYNAMICALLY

The malware uses hashes for both DLL names and the APIs to be used.



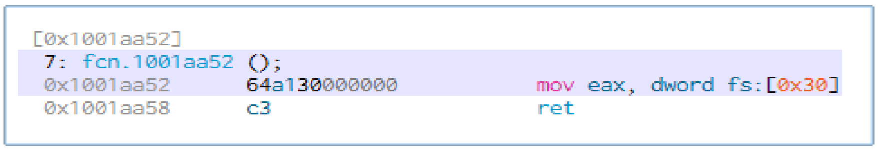
First the matching DLL is found. The below diagram shows how the malware retrieves the DLL names.

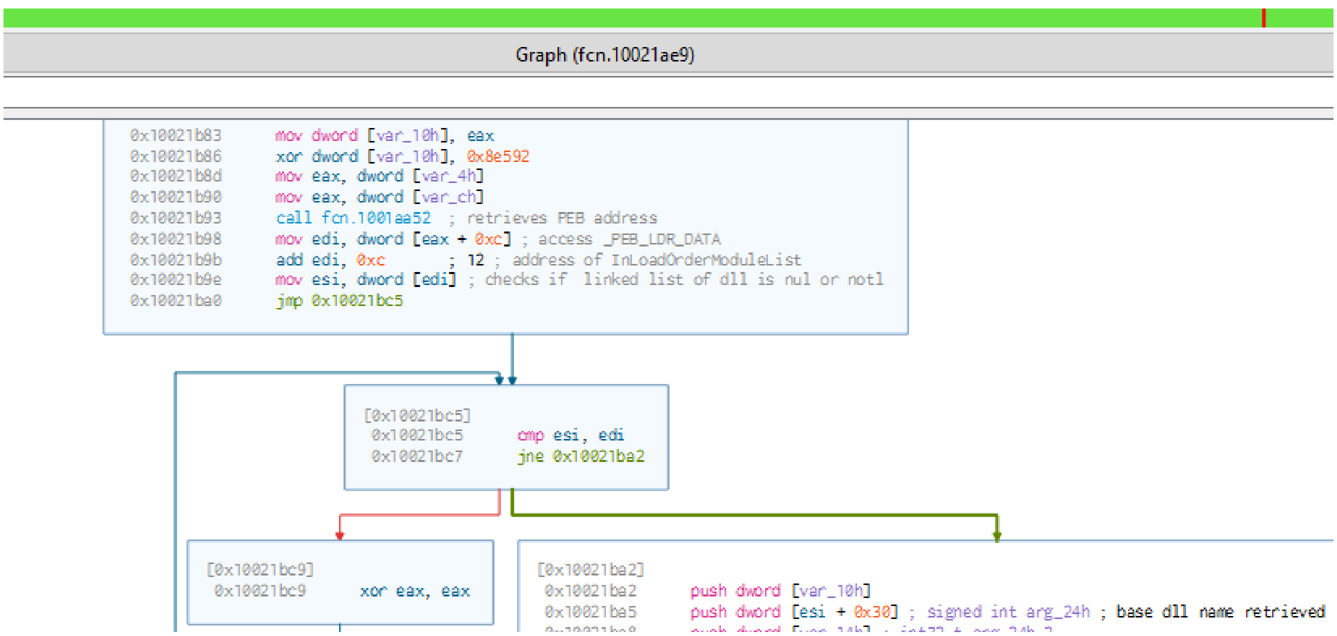


The malware iterates through **LDR_DATA_TABLE_ENTRY** structures for different DLLs to find the DLL name, calculates a hash of the name and matches the hash to the stored hash. If the hash matches the base address of the DLL is retrieved.

The malware then traverses the DLL to the **IMAGE_EXPORT_DIRECTORY** structure of the PE file which holds the fields **Address of Names, Address of Functions, Address of Name Ordinals.**

The Address of Names is an array of pointers, these pointers point to the names of the APIs. The malware calculates the hash based on the names of the APIs and matches it with the stored hash. If there is a match the corresponding address of the function name is retrieved from the Address of Functions array. For more detailed explaination on this check my article here “How Malware Resolves API address dynamically”. The image below gives shows how this functionality is implemented.





The algorithm used to create a hash of a particular API name and compare it later with the stored hash is given below. Since the malware code contains the lot of junk code, the code algorithm here is cleaned up for understanding.

```
int CalculateHashOfAPI(int PointerToAPIName,int junk_variable, int junk_variable)
{
        int temp1 = PointerToAPIName;                    //Pointer to API name
        PointerToAPIName = 0xeadf60;
        PointerToAPIName += 0x5fa9;
        PointerToAPIName = PointerToAPIName / 0x25;
        PointerToAPIName += 0xfffd8b5;
        PointerToAPINa ^= 0x6345b;
        if(*temp1 != 0)
```

```
        {
                do
                {
                        int temp2 = PointerToAPIName;
                        int temp3 = PointerToAPIName;
                        temp3 <<= 0x6;
                        int temp4 = PointerToAPIName;
                        temp4 <<= 0x10;
                        PointerToAPIName = *temp1;          // store character
                        PointerToAPIName += temp3;
                        PointerToAPIName += temp4;
                        PointerToAPIName -= temp2;
                        temp1++;

                }while(*temp1 != 0)

        }
        else
        {
                return PointerToAPIName;                // stores calculated hash
        }

}
```

Acquiring Data and Encryption

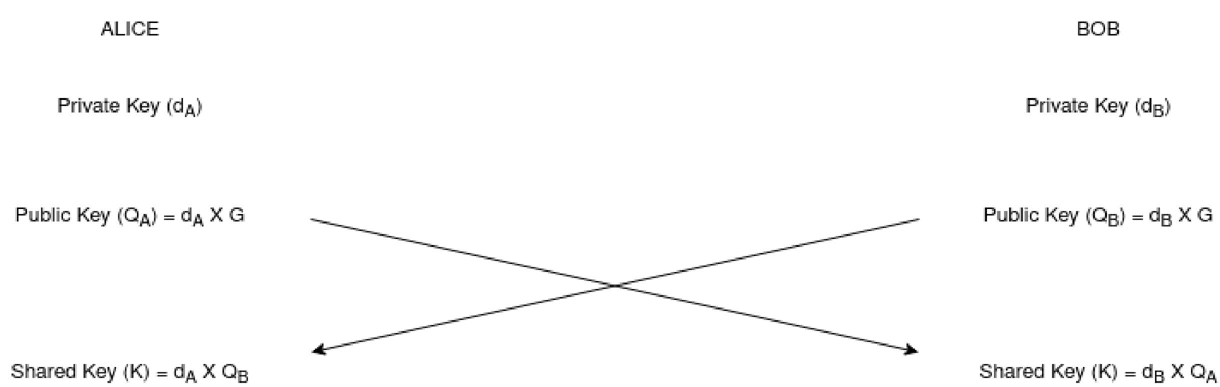
The malware obtains information using various APIs. It encrypts the data and tries to communicate it over to the C2 server. The APIs used for cryptographic operarations are given below.

- <bcrypt.BCryptOpenAlgorithmProvider>
- <bcrypt.BCryptCLoseAlgorithmProvider>
- <bcrypt.BCryptGenerateKeyPair>
- <bcrypt.BCryptFinalizeKeyPair>
- <bcrypt.BCryptExportKey>
- <bcrypt.BCryptImportKeyPair>
- <bcrypt.BCryptSecretAgreement>
- <bcrypt.BCryptDeriveKey>
- <bcrypt.BCryptImportKey>
- <bcrypt.BCryptDestroySecret>
- <bcrypt.BCryptDestroyKey>
- <bcrypt.BCryptHashData>
- <bcrypt.BCryptFinishHash>
- <bcrypt.BCryptDestroyHash>

The Algorithm used by the malware are:

- Elliptic-curve Diffie-hellman (ECDH) :- For key generation and shared secret key.
- Advanced Encryption Standard (AES) :- For encrypting communication with C2 server.
- SHA256 :- For Hashing of data to be send to C2 server.

Below diagram shows the working of the ECDH algorithm where G represents the generator, d the private keys, Q the Public Keys. The operation ‘X’ represents a scalar multiplication. The public key is an (x,y) point on the elliptic curve.



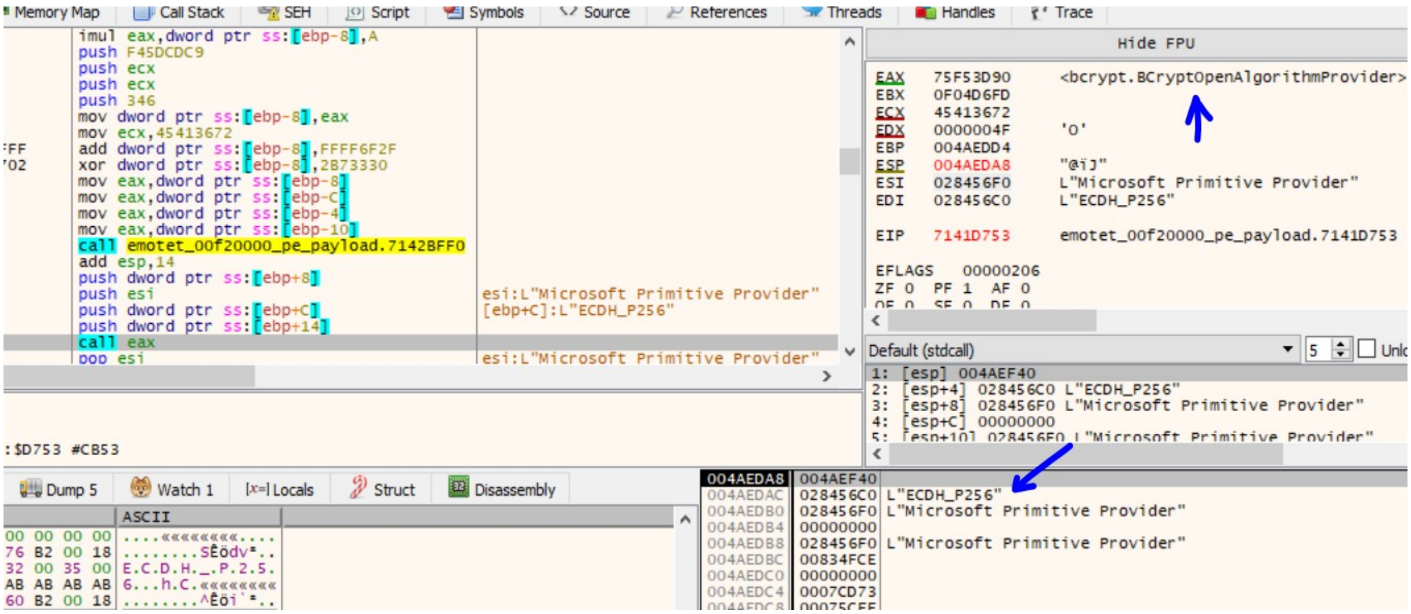
The article wont go into the mathematical details of the algorithm but get an idea in relation to the microsoft APIs we need to know how APIs accomplish the above steps.

Using the **BcryptGenerateKeyPair** API gives us the Public/Private Key pair. The shared key is than derived using the API **BcryptSecretAgreement**. The **BCryptDeriveKey** API is used to obtain the final key. The main task is to **create a shared secret that is common to both the Server and the infected machine in this case. This**

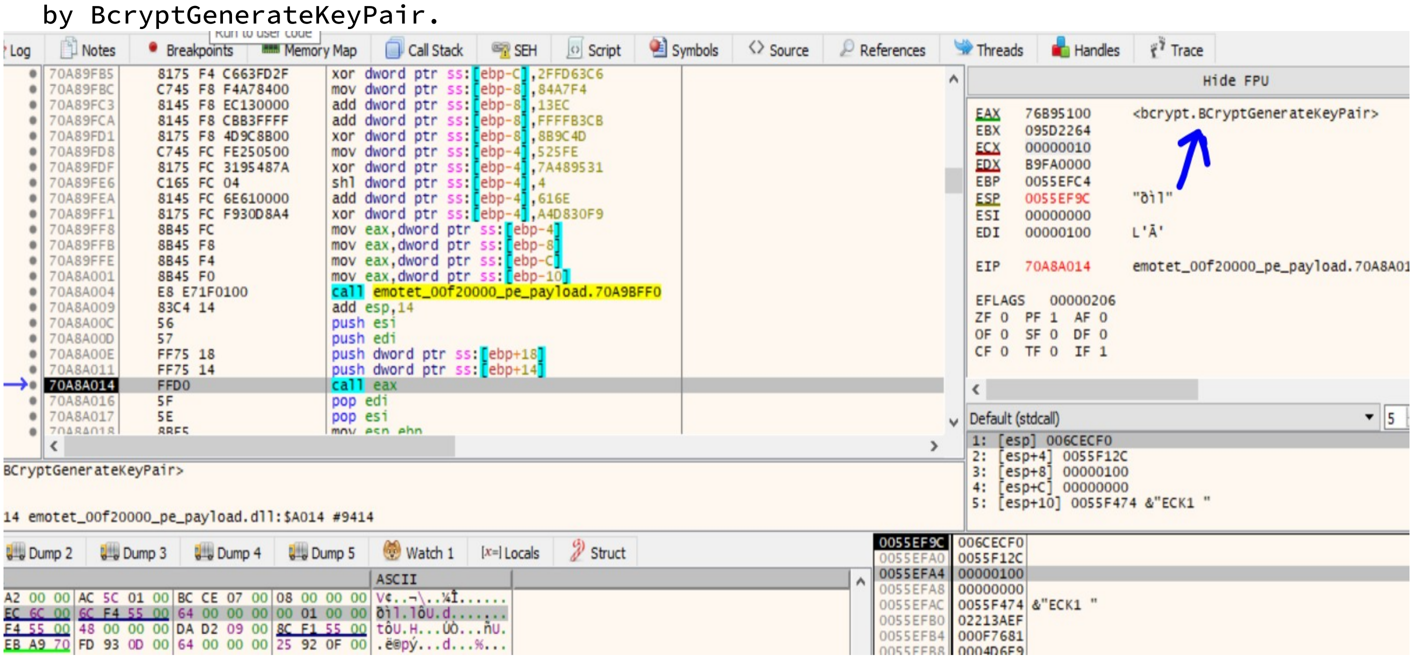
shared secret is then used as input to the AES algorithm, which than outputs the final symmetric key that will be used for encrypting messages.

ENCRYPTION OPERATIONS

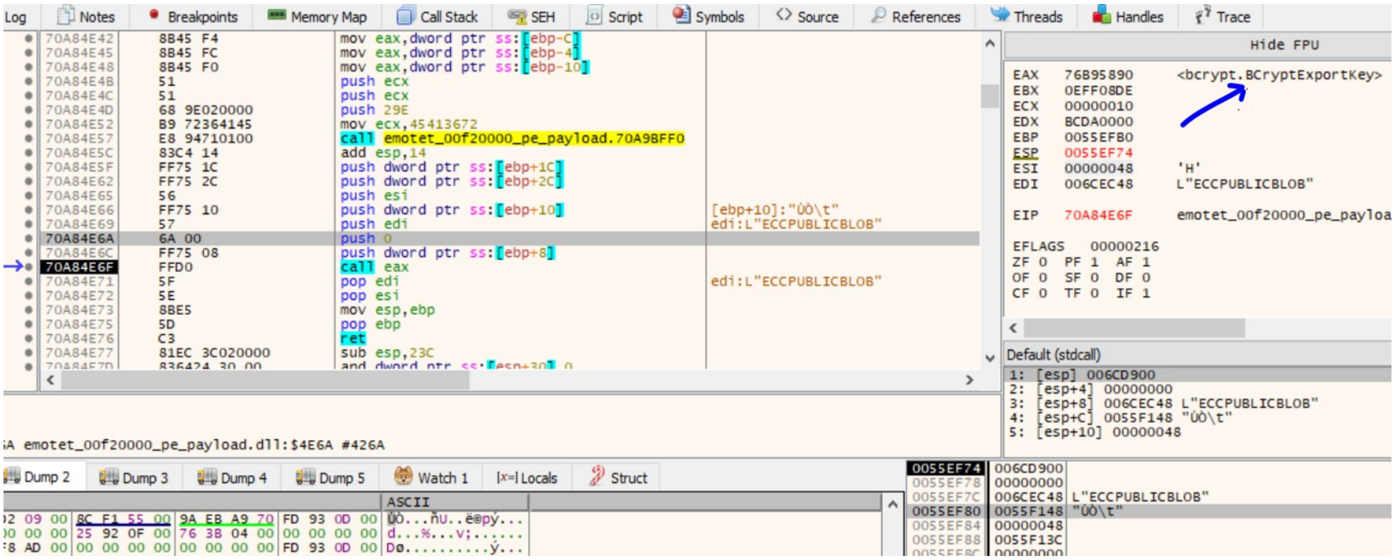
BCryptOpenAlgorithmProvider API initializes a CNG(Cryptography API: Next Generation) provider. The algorithm used by emotet malware in this case is ECDH_P256 which is Elliptic-curve Diffie-hellman algorithm.



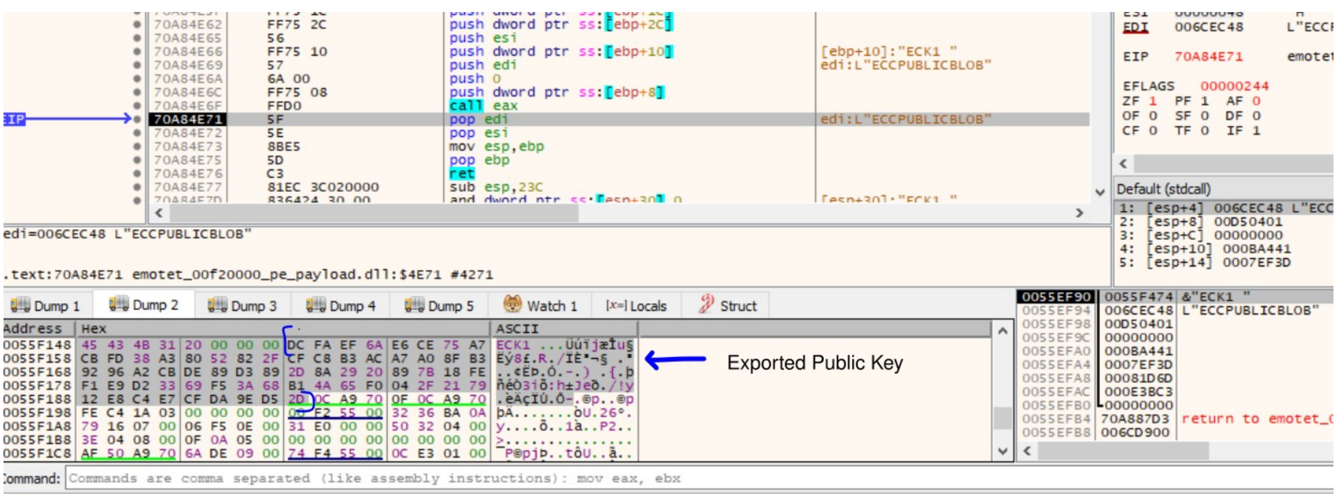
The handle of that is than passed to the **BCryptGenerateKeyPair** which creates a public/private key pair. **BCryptFinalizeKeyPair** API finalizes the key pair generated by BcryptGenerateKeyPair.



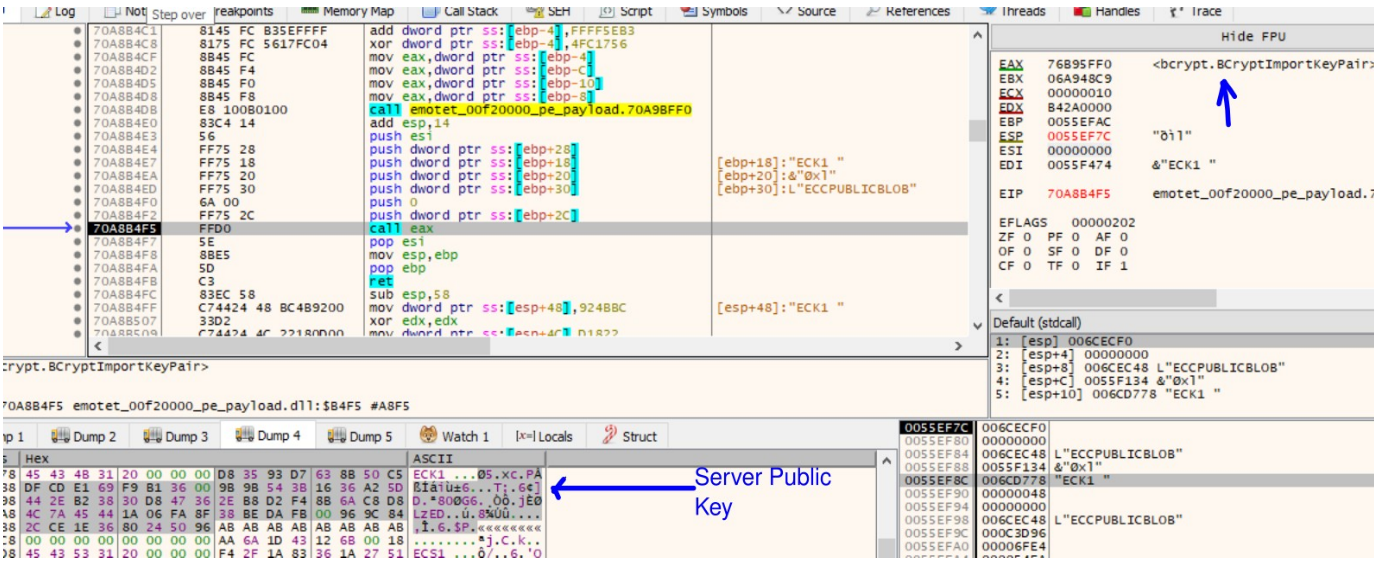
BCryptExportKey API is then used to export the Public key from the key pair to a **BCrypt_ECCPUBLIC_BLOB structure.** This public key can now be sent to the malicious server.



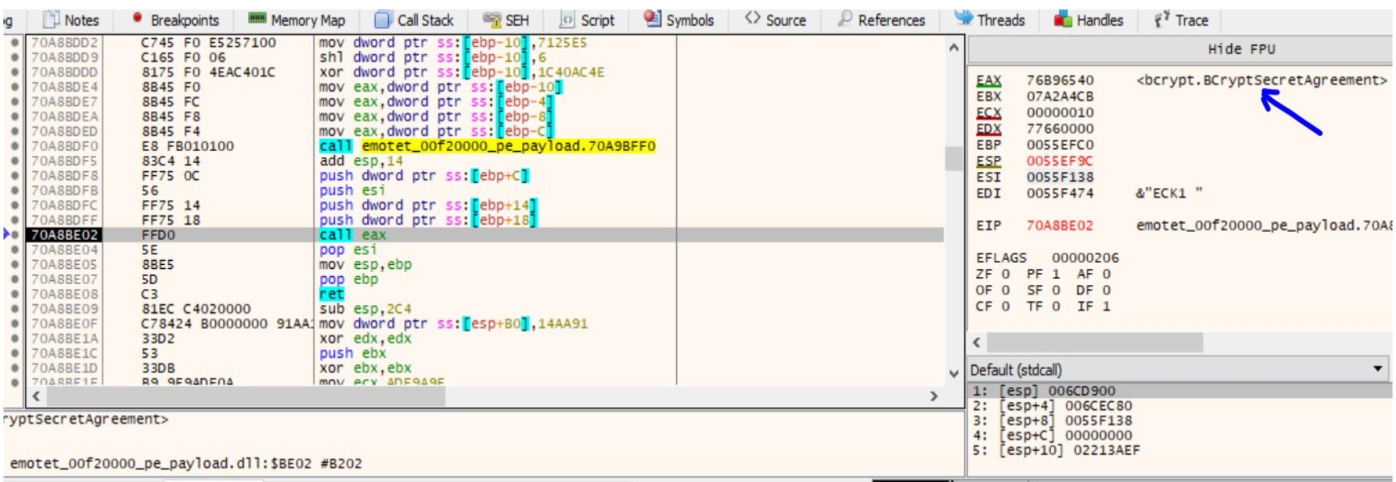
Exported Public Key inside the structure can be seen in the image below.



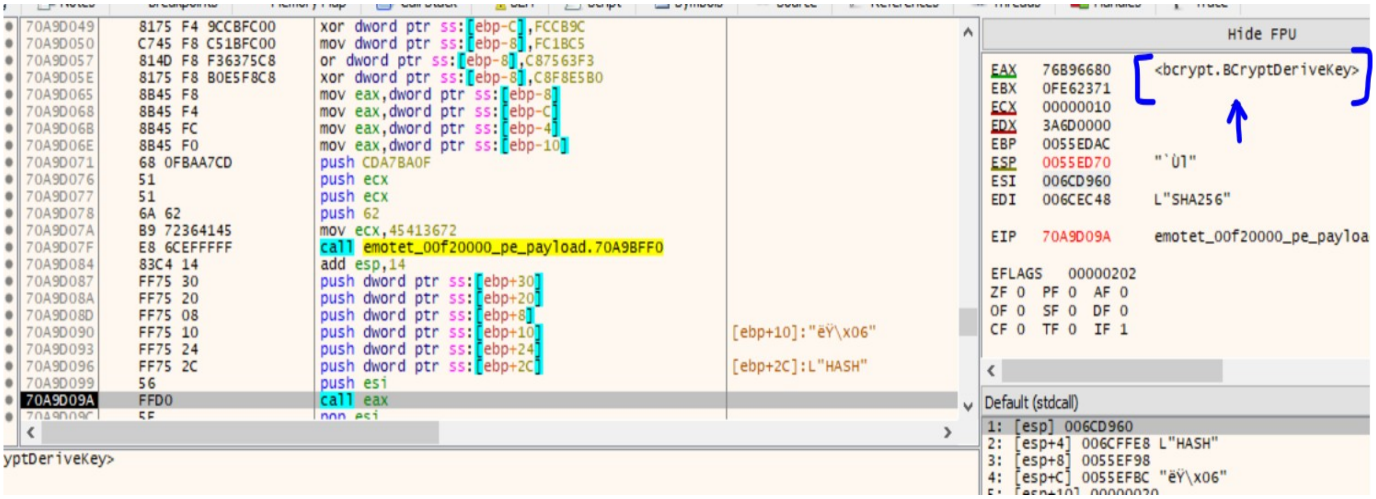
The Public Key of the Server is embedded in the malware inside a BCrypt_ECCPUBLIC_BLOB structure. **BcryptImportKeyPair** API is used to obtain a handle to it.



The handle of the public/private key pair obtained using BcryptGenerateKeyPair and the handle of the public key of the Server obtained in the previous step is passed to the **BcryptSecretAgreement** API. This returns a handle to the shared secret.



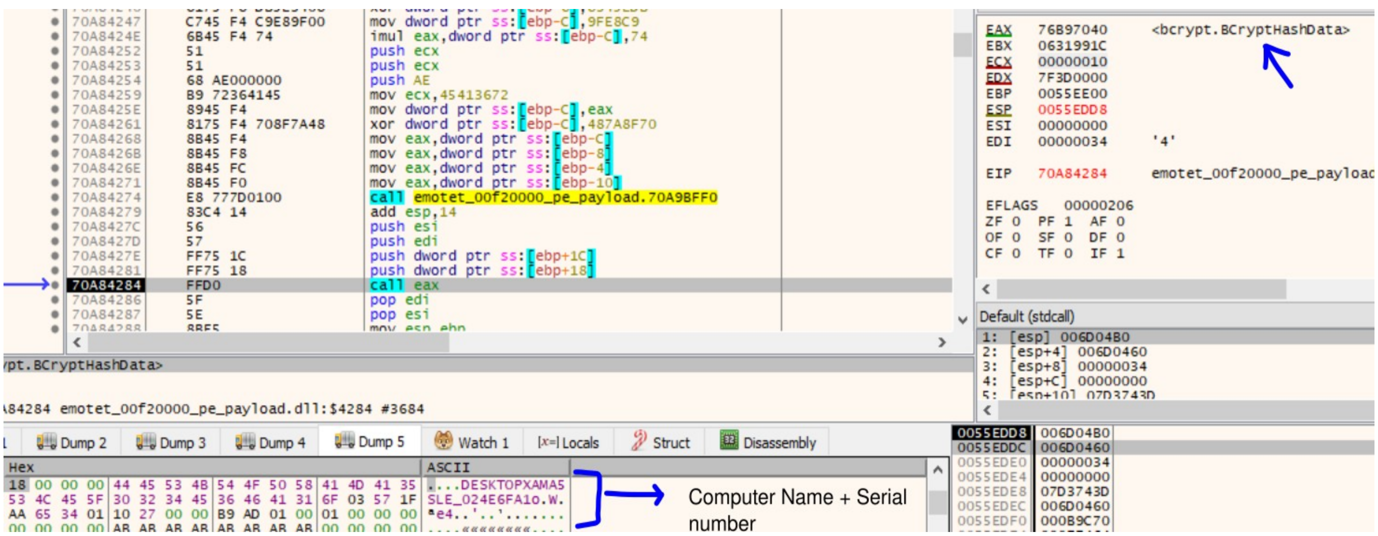
The **BCryptOpenAlgorithmProvider** is then called to initialize the AES CNG provider. The handle obtained previously of the shared secret is passed to the **BCryptDeriveKey** API to obtain a final **symmetric(same for both malware and server) key of size 32 bytes** ie. 256 bit key.



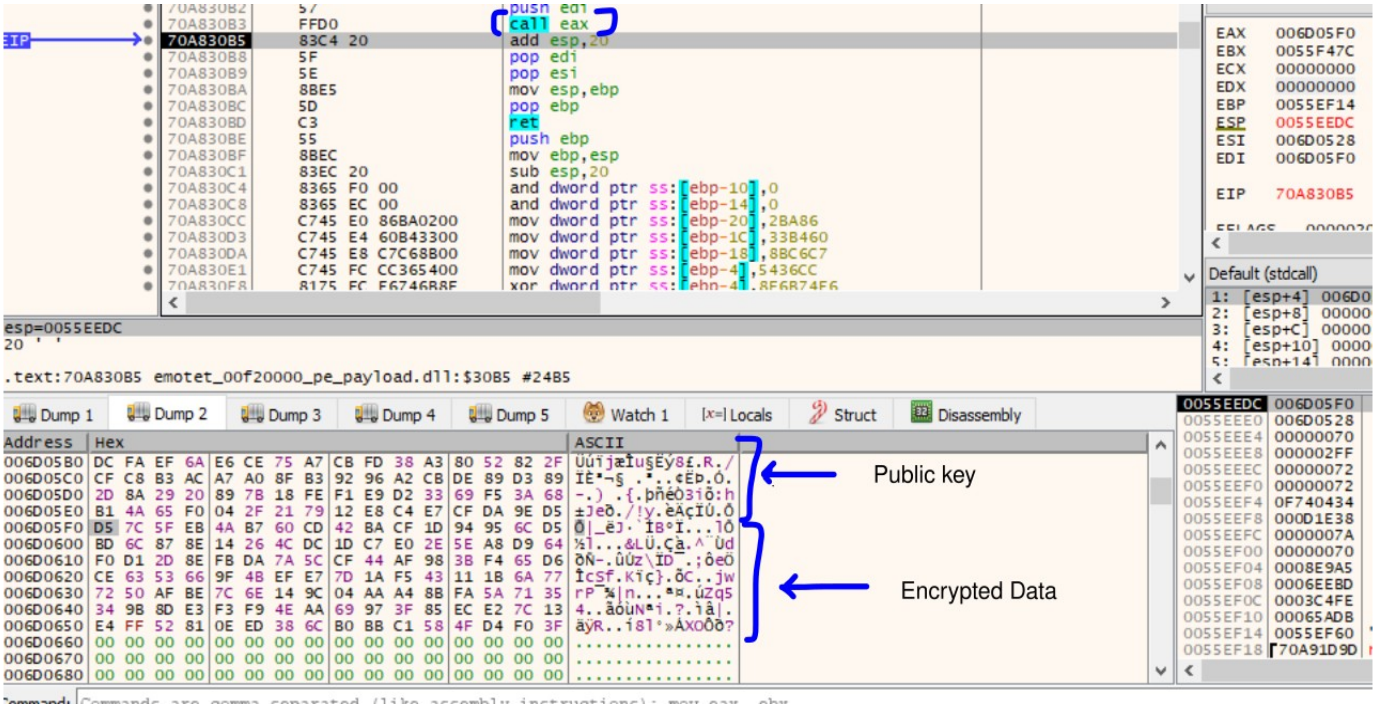
BCryptImportKey is than used to obtain a handle to this key. This key will later be used to encrypt the messages sent to the server. All the other intermediate keys are destroyed using **BCryptDestroySecret** and **BcryptDestroyKey**.

Before the Data is to be sent. The hash of the data is calculated, in this case the data is the **Computer name obtained using the API GetComputerNameA** and the **serial number of the C Drive**, that is retrieved using the **GetVolumeInformation** API. Both are concatenated and sent to the server later after encryption.

The BCryptAlgorithmProvider is called to initialize the SHA256 CNG provider. **BCryptCreateHash** API is called to create a hash object and **BCryptHashData** API is used to hash the data as seen below.



The ComputerName and the hash are then encrypted using the **BCryptEncrypt** API.

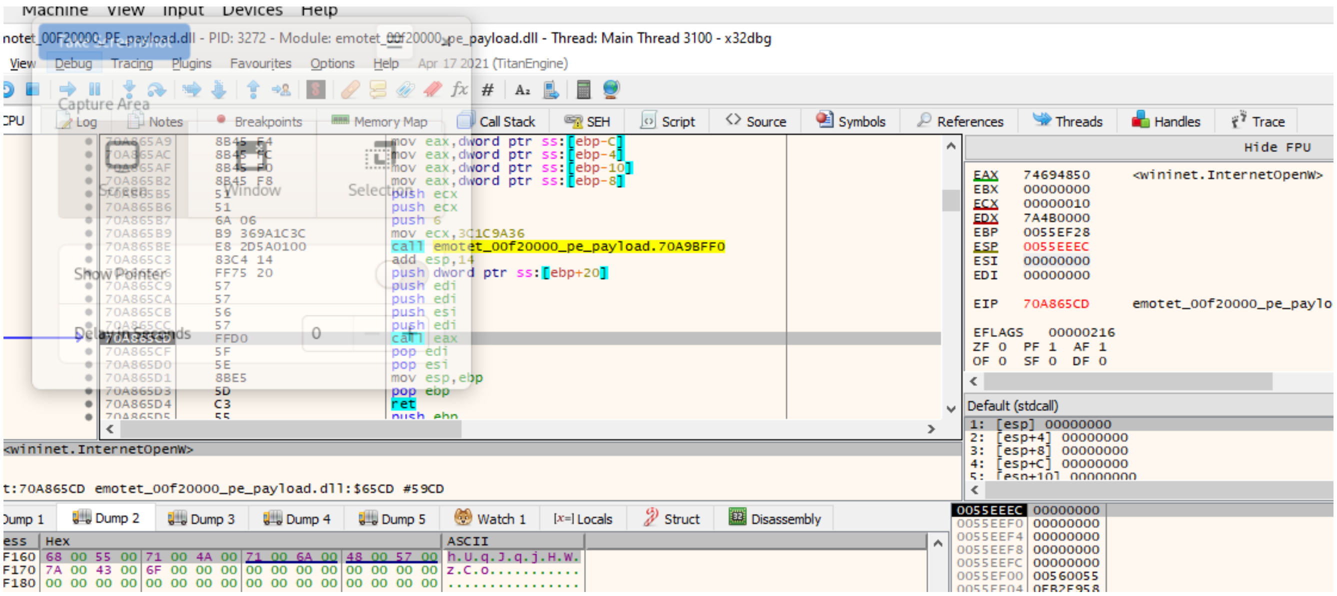


COMMUNICATION WITH C2 SERVER

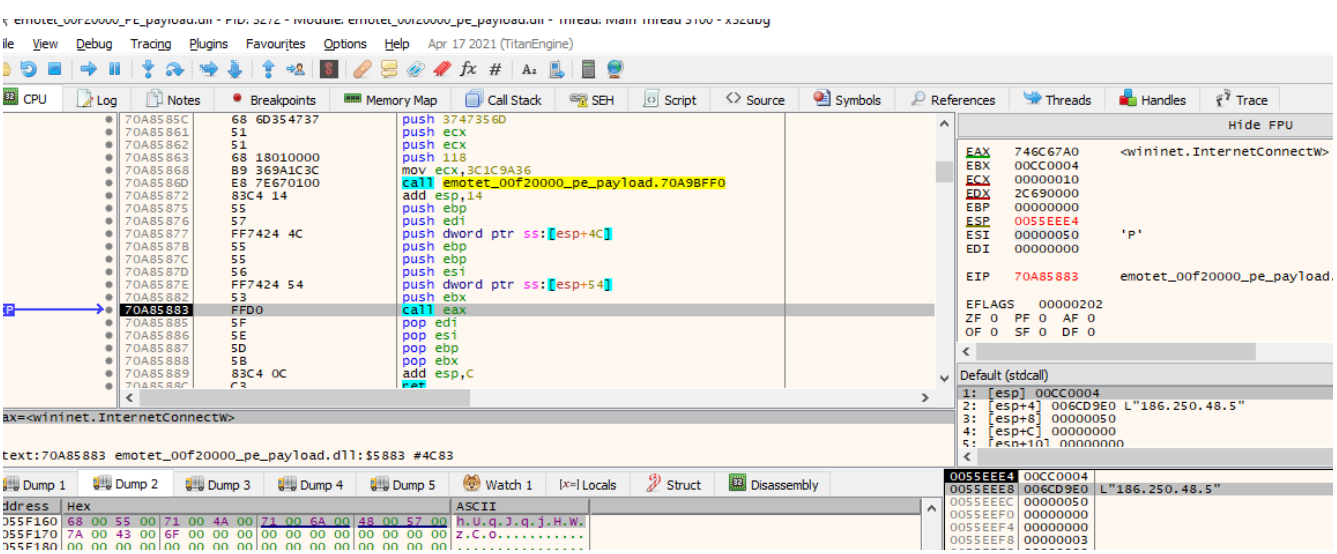
The data after being converted is prepared to be sent to the server. The sequence of API's to establish connection to the server are:-

1. wininet.InternetOpenW
2. wininet.InternetConnectW
3. wininet.HttpOpenRequestW
4. wininet.InternetSetOpitionW
5. wininet.InternetQueryOptionW
6. wininet.HttpSendRequestW

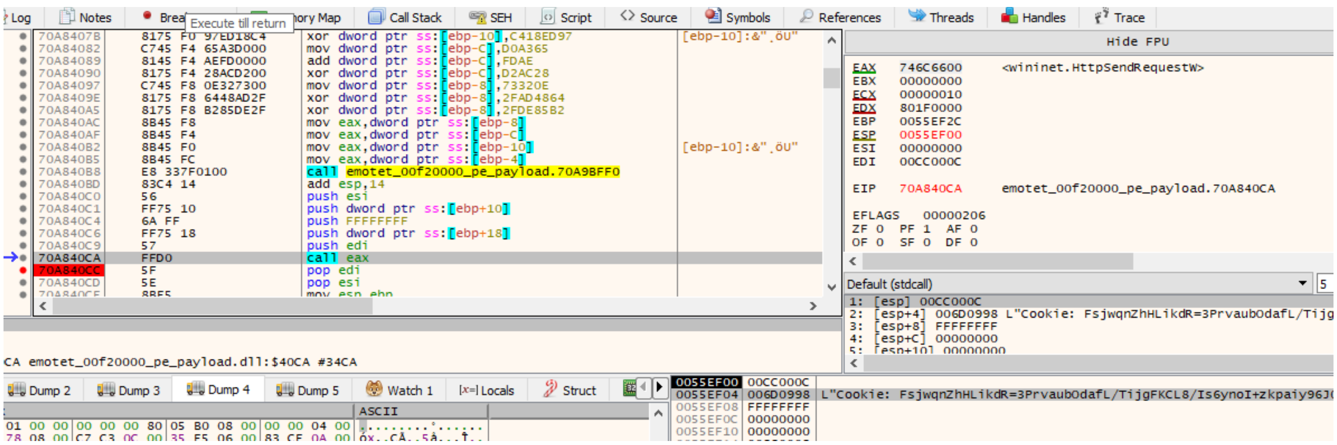
The malware first uses the **InternetOpenW** api to get a handle that is later used by other wininet apis.

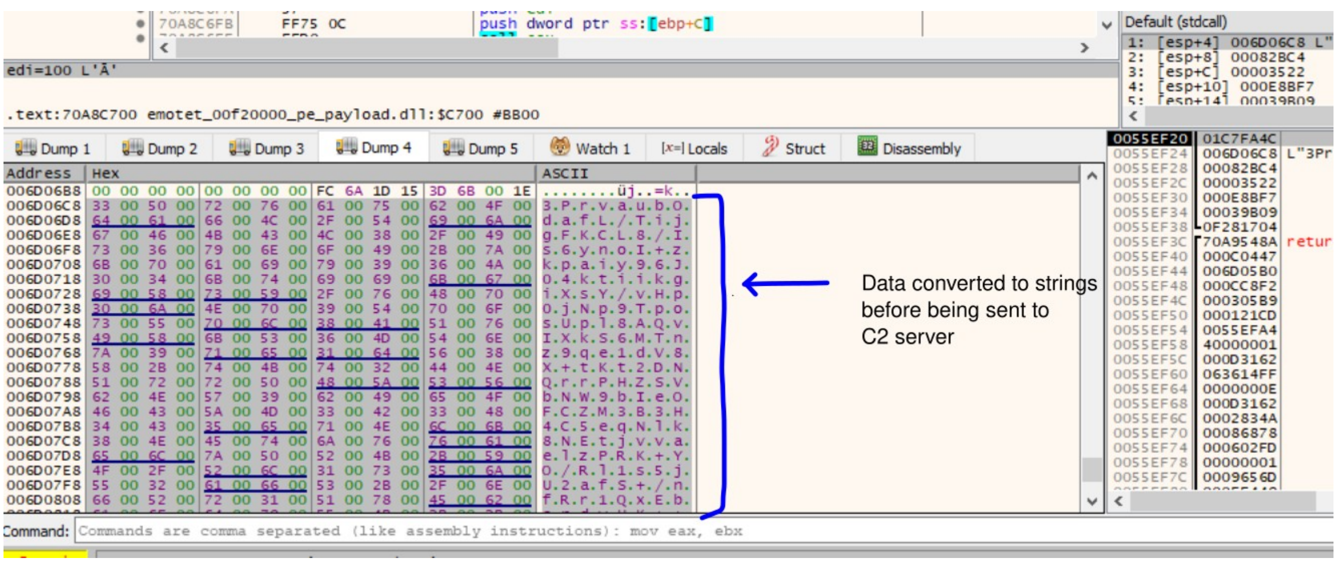


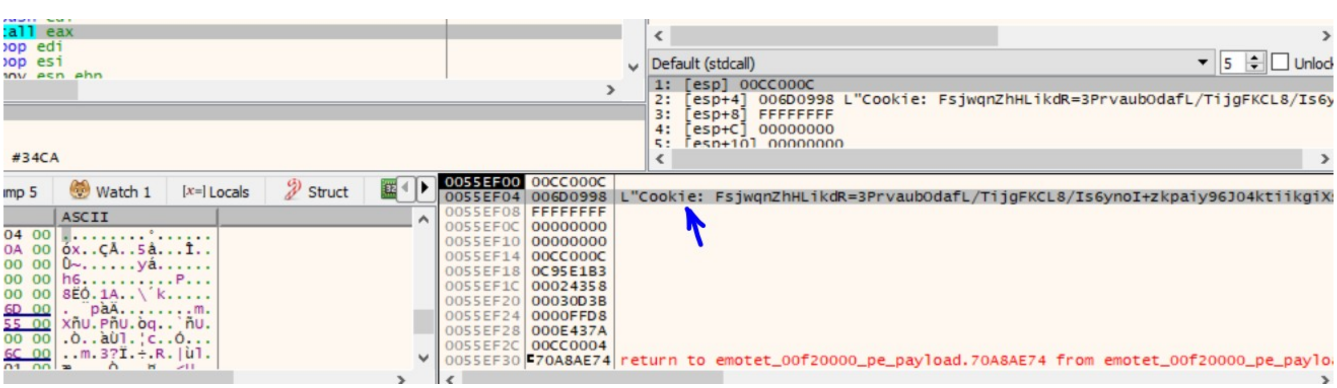
The malware then calls the **InternetConnectW** api to start a session for the address 186.250.48.5 over port 50.



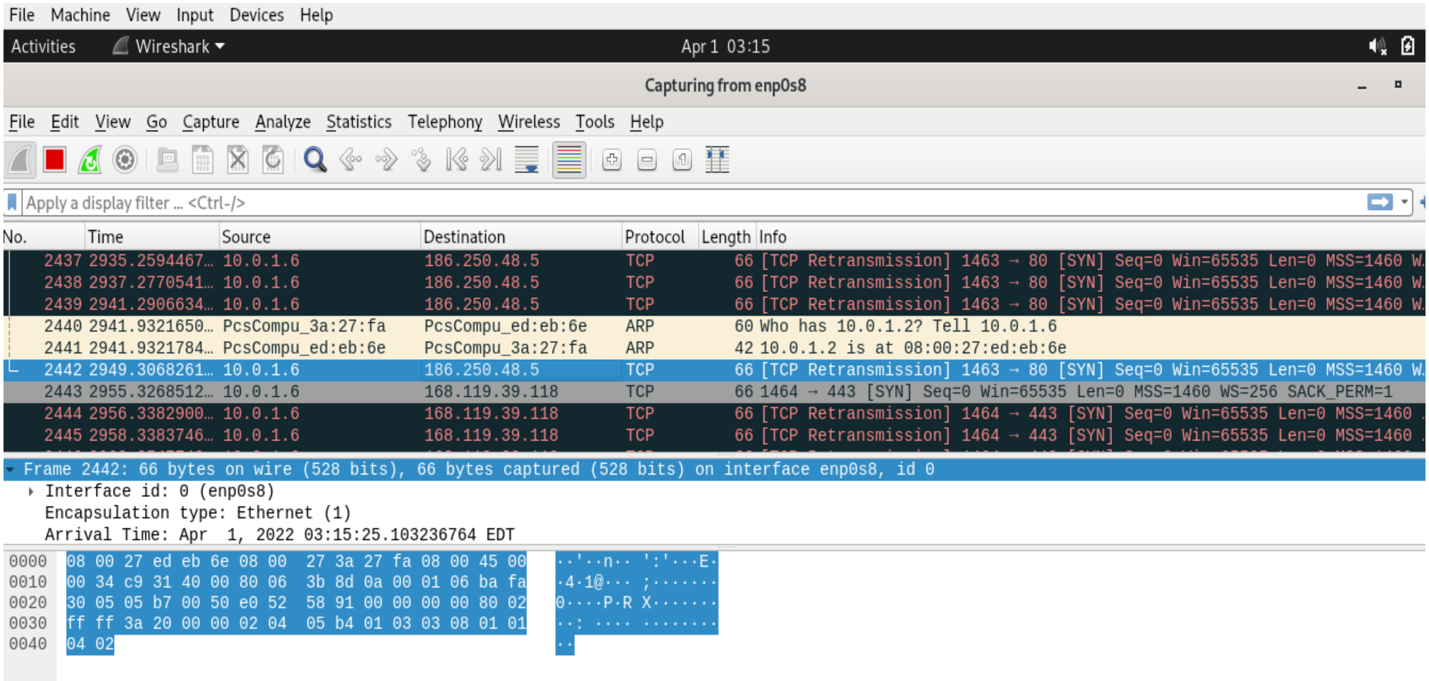
HttpOpenRequestW api is then used to create an Http request handle. The Get method is used in this case and the object name is **hugJqjMWzCo. HttpSendRequestW** method is used to send the request to the Malicious server. The Http additional header is where the malware stores and sends the data. In this case it is the data previously seen i.e **the Public key of the machine + Encrypted Data**. Everything is then converted to strings using the API **CryptBinaryToStringsW**. Which is then appended to the header as the field named **Cookie** in this case.







The connection request from the malware to the C2 server can be seen below.



INDICATORS OF COMPROMISE

- **Host Based IOCs**
  - File:- ae5de878deeb48308865377d6a71a769dbf74a06985fa7be19ebdb7a85ed316b

- **Network Based IOCs**
  - 186.250.48.5
  - 168.119.39.118
  - 185.168.130.138
  - 190.90.233.66
  - 159.69.237.188
  - 54.37.228.122
  - 93.104.209.107
  - 185.148.168.15
  - 198.199.98.78
  - 87.106.97.83
  - 195.77.239.39
  - 37.44.244.177
  - 54.38.242.185
  - 185.184.25.78
  - 116.124.128.206
  - 139.196.72.155
  - 128.199.192.135
  - 103.41.204.169
  - 78.47.204.80
  - 68.183.93.250
  - 194.9.172.107

INDICATORS OF ATTACK

- **Data exfiltration**
  - Acquires drive serial number, Computer name and Operating System Version etc to be sent to the C2 server.
- **C2 Communication**
  - Tries to connect to C2 server with periodic connection request.
- **Stealth**
  - Almost all APIs address are resolved dynamically
  - Junk code usage