

# Ransomware Alert: Pay2Key

research.checkpoint.com/2020/ransomware-alert-pay2key/

November 6, 2020



November 6, 2020

## Introduction

Over the past week, an exceptional number of Israeli companies reported ransomware attacks. While some of the attacks were carried out by known ransomware strands like REvil and Ryuk, several large corporations experienced a full blown attack with a previously unknown ransomware variant names **Pay2Key**.

As days go by, more of the reported ransomware attacks turn out to be related to the new Pay2Key ransomware. The attacker followed the same procedure to gain a foothold, propagate and remotely control the infection within the compromised companies.

The investigation so far indicates the attacker may have gained access to the organizations' networks some time before the attack, but presented an ability to make a **rapid move of spreading the ransomware within an hour to the entire network**. After completing the infection phase, the victims received a customized ransom note, with a relatively low demand of 7-9 bitcoins (~\$110K-\$140K).

The full scope of these attacks is still unraveling and is under investigation; but we, at Check Point Research and the Incident Response teams, would like to offer our initial analysis of this new ransomware variant, as well as to provide relevant IOC's to help mitigate possible ongoing attacks.

### Key findings:

1. Previously unknown ransomware dubbed Pay2Key, carries targeted attacks against Israeli companies
2. Initial infection is presumably made through RDP connection
3. Lateral movement is made using psexec.exe to execute the ransomware on the different machines within the organization.
4. Special attention was given to the design of the network communication, in order to reduce the noise a large number of encrypted machines may generate while contacting the Command and Control servers.
5. The encryption scheme is solid – using the AES and RSA algorithms.

## Attacks Timeline

During the last days, we were able to obtain bits and pieces of information as well as various forensics artifacts from Israeli Incident Response teams, indicating that a new ransomware strain is being deployed against Israeli corporations (perhaps exclusively).

Combining these elements, we were able to bring together a partial image of the attacks as they unfolded:

- 2020-06-28 – The attacker created a KeyBase account by the name of “pay2key”
- 2020-10-26 – First ransomware sample compilation date
- 2020-10-27 – Second ransomware sample compilation date
- 2020-10-27 – First Pay2Key sample uploaded to VT and compiled on the same day – may indicate its first appearance in the wild.
- 2020-10-28 – Second ransomware sample uploaded to VT – Indicating a possible attacked organization.
- 2020-11-01 – Third sample compilation date
- 2020-11-01 – The first reported attack (Sunday; working day in Israel)
- 2020-11-02 – The second reported attack

The Pay2Key propagation appears to be conducted as follows:

1. Right after midnight, the attackers connected to a machine on the targeted network most probably **via RDP**.
2. A machine is defined as **Pivot / Proxy** point within the network, likely by using a program named “ConnectPC.exe”. All outgoing communication between all ransomware processes within the network and the attacker’s C&C server will be going through this proxy from this point on.
3. The attacker used psexec.exe to execute “Cobalt.Client.exe”, which is the Pay2Key ransomware itself, on different machines within the organization.

## New Ransomware

---

Analyzing **Pay2Key** ransomware operation, we were unable to correlate it to any other existing ransomware strain, and it appears to be developed from scratch.

Only a single engine on VirusTotal detected the uploaded ransomware samples as malicious, even though the ransomware does not use a Packer or protection of any kind, to hide its internal functionality.

Numerous compilation artifacts point to the fact that internally, this ransomware is in fact named **Cobalt** (not to be confused with Cobalt Strike).

While the identity of the attacker is unknown, inconsistent English wording within the various strings found in the code, as well as the ones we observed in the Log file, suggests that the attacker is not a native English speaker.

## Ransom Demand

---

After successful encryption, the ransomware drops a ransom note to the system, customized to the targeted corporation in the form of `[ORGANIZATION]_MESSAGE.TXT`. The ransom amount ranges between 7 and 9 Bitcoins, among the ransom notes we observed.

# PAY2KEY

HELLO ██████████ USERS!

Congratulations!

Your entire network and all your informations such as computers/ employees information/ users folders/ servers/ file-servers/ applications/ databases/etc... in your network has been successfully encrypted!

Some of your important information dumped and ready to leak, in case we can't make a good deal!

Don't modify encrypted files or you can damage them and decryption will be impossible!

Don't try unofficial decryptors to recover your files or you can damage them and decryption will be impossible!

There is only ONE possible way to get back your files! Pay and contact to receive your special decryptor!

You should pay 7 BTC to receive official decryptor and easily recover your files. ██████████ special decryptor is now ready and waiting for your payment, let's do it!

You can send 4 random files from any computers and receive decrypted data, just as a proof that works!

Your UserID IS: ██████████

Your Network ID ██████████

| NOTICE |

Offer available until 11/08/2020. If you do not pay on time, price will be doubled!

Wallet: ██████████

E-mail:

pay2key@tuta.io

pay2key@pm.me

Keybase:

Pay2Key

Figure 1: Pay2Key ransom note – even the ASCII-art is customized per organization

Worth mentioning, that although the ransom note informs the victims for data breach, like other double extortion ransomwares do, we have yet to find any evidence that supports it.

## Pay2Key

One interesting thing to note is that the Keybase account used by the attacker to chat with their victims has the same logo of the Pay2Key EOSIO smart contract system. A possible explanation is the fact that when searching “pay2key” in Google images, this is the first result.



**pay2key**

Pay2Key, no more...  
Internal networks.

2 devices

Chat with pay2key

Your conversation will be end-to-end encrypted.

Figure 2: Pay2Key Keybase.io profile

## Technical Analysis

### Initial Access

Our analysis of the attack by Pay2Key focused on the binary of the ransomware itself since some of the previous stages in the attack were not accessible to us. The attack, as we mentioned earlier, started by manually accessing one of the machines on the victim's network, likely via RDP. The attacker copied and created multiple files on the machine, including:

- Cobalt.Client.exe – Pay2Key ransomware
- Config.ini – A configuration file that specifies “Server” and “Port”
- ConnectPC.exe – Pivot / Proxy server

After the creation of these files on the infected machine, the attackers execute `ConnectPC.exe`. Then, they copied or downloaded the `PsExec` utility and used it to remotely execute the ransomware on other machines in the organization. In order to work properly, the ransomware requires a config file to be located in the same working directory. Thus, `Config.ini` is required to be dropped in the victim's computer along with `Cobalt.Client.exe`. In the cases we've seen, the Pay2Key ransomware was executed from paths of this template: `C:\Windows\Temp\[organization-name]tmp\Cobalt.Client.exe`

## Configuration

The artifacts we were able to put our hands on are the ransomware, `Cobalt.Client.exe`, and the configuration file. The configuration file is a very simple INI file that contained two entries — Server and Port. To our surprise, the Server wasn't an external command and control server, but rather the IP of the initial infected machine. Thus, we believe that the original machine was using `ConnectPC.exe` as a utility to relay communication from victims inside the organization to the external control server. This approach increases the chance that the different machines will be able to communicate because internal communication is more likely to be allowed. It also decreases the chances that the address of the command and control will be revealed by analysts as there is only one machine in the organization that knows of it.

The configuration file that was used in the attack looked like this:

```
[Config]
Server = <internal IP address>
Port = 5050
```

If the ransomware was executed with `--config [path]` as a command-line argument, it will read the configuration file from the path specified in the argument.

## The Ransomware

The Pay2Key ransomware is written in C++ and compiled using MSVC++ 2015. It heavily relies on Object-Oriented Programming and uses well-designed classes for its operation. It also makes use of 3rd-party libraries like the popular libraries of Boost. Luckily, the ransomware was not stripped and it contained a decent amount of debug logs as well as rich RTTI information.

```
$ diec Cobalt.Client.exe
PE: compiler: Microsoft Visual C/C++(2015 v.14.0)[-]
PE: linker: Microsoft Linker(14.0, Visual Studio 2015 14.0*)[EXE32,console]

$ rabin2 -I Cobalt.Client.exe | grep "compiled\|pdb"
compiled Mon Oct 26 12:37:49 2020
dbg_file F:\2-Sources\21-FinalCobalt\Source\cobalt\Cobalt\Cobalt\Win32\Release\Client\Cobalt.Client.pdb
```

Upon execution, Pay2Key is reading the Server and Port keys from the configuration file. If a configuration file was not found in the current working directory and wasn't supplied in the command line arguments, the ransomware will write “no config file found” to a file at `.\Cobalt-Client-log.txt`. This log file will be used extensively by the ransomware during its execution. Newer versions of the ransomware are making sure to remove this log file from the disk. The full list of supported log messages can be found in the appendix section of this article.

It then initializes the main class of the program, `Cobalt::DataProcessing::RansomwareEngine`, followed by initialization of other important classes that are responsible, among other things, for communication, message handling, managing files and encryption.

Pay2Key generates a pair of RSA keys and sends the public key to the server over raw TCP. The keys will be used to set up secure communication between the ransomware and the server. After sending the key, the ransomware will wait for messages from the server. These messages are parsed and handled by a custom Message Handler.

## Supported Messages

Message_ID	Message_Name	Notes
0	PublicKey	Receive the server's public key
1	Identification	Send to the server the IP address, the MAC address and the hostname.
2	Config	Receive a configuration from the server. The configuration is a very important aspect in the ransomware as it contains valuable information such as a list of file extensions to encrypt, the name of the victim's organization, the ransom note, the extension of the encrypted file, and more.
3	ExceptionMessage	
4	SessionKey	Receive unique session key from the server

Message_ID	Message_Name	Notes
5	JobFinished	Announce that the encryption job is finished
6	Abort	Stop the execution
7	GetClientList	N/A
8	ClientList	N/A
9	GetClientInformation	Send, upon request, status of different tasks like the encryption task.
10	ClientInformation	Send status of different tasks like the encryption task.
11	Acknowledge	
12	GetIdentification	Send to the server, upon request, the IP address, the MAC address and the hostname.
13	None	N/A

The ransomware uses dedicated classes to handle different messages. We noticed some of these handlers and message types are not implemented in the binaries we have, and some of them aren't even checked by the message handler manager. This is another indication that the ransomware is under active development. The missing messages, perhaps, can be part of code that is now removed, or alternatively — code that wasn't implemented yet.

During the reverse engineering process of the ransomware samples, we utilized the fact they contain rich RTTI data. Such information helps us understand the role of different pieces of the code and the relationship between them. With the help of [Cutter](#), we were able to inspect the different classes elegantly.

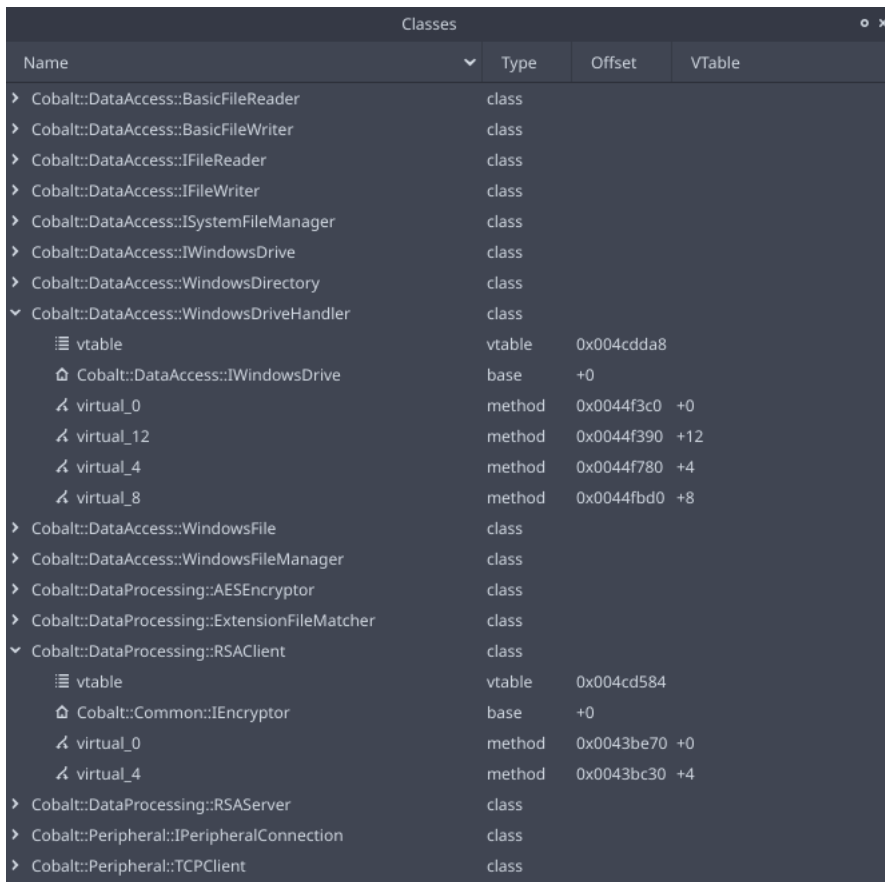


Figure 3: RTTI is parsed By Cutter and shown

in the Classes window

The most interesting message type is "Config" as it contains information that the ransomware use during the infection. Analyzing the ransomware without the information from this message results in constant fallbacks to default configurations. Among other things, the configuration message contains the list of file extensions to encrypt, the file extension for encrypted files (default `.enc`), the name for the ransom note file (default to "salam" as in `SALAM_MESSAGE.TXT`), and the ransom note.

It is interesting to mention, that both the file name and the ransom note specified the name of the infected organization. The attackers even went all the way to generate an ASCII-Art with the name of the infected organization. The file name of the ransom note from the config will have the following template `[ORGANIZATION]_MESSAGE.TXT`. In the incidents we analyzed, the extension that was received from the server

was `.pay2key`, but it could have been anything else as the ransomware is flexible enough.

At the end of the encryption process, Pay2Key will also terminate the MS SQL service using the following command `net stop mssqlserver > nul` in order to release the files locked by the service. It might also replace the wallpaper of the victim, but we did not see this happening on the machines we analyzed.

## Evolution

---

We analyzed multiple samples in a small period of time and we noticed several improvements in them. This means that Pay2Key is under active development and the developers update it with more features. For example, in the latest version of the ransomware, we noticed that the attackers added a Self Killing mechanism, in addition to a new command-line argument `--noreboot`.

The new “housekeeping” mechanism is responsible for removing the files created by the attacker and restarting the machine.

```
void __cdecl __noreturn ic_houseKeepingAndReboot(int a1, int a2)
{
    int v2; // edx
    int v3; // [esp+0h] [ebp-18h]
    int a2a; // [esp+4h] [ebp-14h]
    void *v5; // [esp+8h] [ebp-10h]
    int v6; // [esp+14h] [ebp-4h]

    g_allowShutdown = 1;
    ic_writeToLog("Deleting Myself");
    ic_Cobalt::Communication::ExceptionMessag::init_0(v2, &a2a, v3, a2a, v5);
    v6 = 0;
    ic_writeToLog("Restarting");
    ic_pingAndSelfDelete();
    if ( g_rebootAllowed )
        ic_runCmdCommand((int)"Shutdown /r /f /t 0");
    ic_writeToLog("Application Closed");
    DeleteFileW(L"\\.\\Cobalt-Client-log.txt");
    DeleteFileW(L"\\.\\Config.ini");
}
```

Figure 4: The function that is responsible for removing the

ransomware and its files and restarting the system

## Encryption

---

As standard for ransomware, a hybrid of symmetric and asymmetric cryptography is used for file encryption — using the AES and RSA algorithms. The C2 server supplies an RSA public key at runtime during communication. This implies that this ransomware doesn't encrypt offline – if there's no internet connection, or the C2 is down, then no encryption will occur. Recent years have seen some ransomware that aggressively use cryptographic primitives to put the onus of creating successful contact with the operations on the victim (e.g. by embedding the server public key in the executable), and at the time this was considered a technical improvement, but evidently the authors here prefer the classic flavor.

```

mov     [ebp+var_14], 0
push   ebx                ; phKey
push   1                  ; dwFlags
push   CALG_RC4           ; AlgId
push   dword ptr [edi+4] ; hProv
call   ds:CryptGenKey
mov     [ebp+var_20], 0
mov     [ebp+var_1C], 0
mov     [ebp+var_18], 0
lea    eax, [ebp+pdwDataLen]
mov     [ebp+var_4], 0
push   eax                ; pdwDataLen
push   0                  ; pbData
push   0                  ; dwFlags
push   1                  ; dwBlobType
push   dword ptr [edi+0Ch] ; hExpKey
push   dword ptr [ebx]    ; hKey
call   ds:CryptExportKey
test   eax, eax
jz     short loc_43BA83

push   [ebp+pdwDataLen]
call   unknown_libname_69 ; Microsoft VisualC 14/net runtime
push   [ebp+pdwDataLen] ; Size
mov     esi, eax
push   0                  ; Val
push   esi                ; void *
call   _memset
add    esp, 10h
lea    eax, [ebp+pdwDataLen]
push   eax                ; pdwDataLen
push   esi                ; pbData
push   0                  ; dwFlags
push   1                  ; dwBlobType
push   dword ptr [edi+0Ch] ; hExpKey
push   dword ptr [ebx]    ; hKey
call   ds:CryptExportKey
test   eax, eax
jnz   short loc_43BACD

```

Figure 5: A symmetric RC4 key is generated and

encrypted using an RSA public key.

One unusual thing to point out is the use of RC4 for some (not all) of its cryptographic functions. RC4 is a stream cipher, and is easier to misuse in catastrophic ways; it is usually popular among malware authors for its ease of implementation, but here the authors actually used a third party implementation (via the Windows API). This may be the first ever time we've seen malware authors essentially say "we have the whole world of cryptography at our fingertips, third party libraries for everything, powerful symmetric crypto as far as the eye can see. Let's pick... RC4". But, again, for this to really matter would require some sort of subtle error when invoking the cipher, and we were not able to pinpoint one.

```

push   28h ; '('          ; dwFlags
push   1          ; dwProvType
push   dword ptr [edi+1Ch] ; szProvider
push   offset a_pippo_container ; "Pippo Container"
push   esi        ; phProv
call   ds:CryptAcquireContextW
test   eax, eax
jz     short loc_43BE8B

loc_43BE65:
add    edi, 18h
push   edi        ; phUserKey
push   1          ; dwKeySpec
push   dword ptr [esi] ; hProv
call   ds:CryptGetUserKey
test   eax, eax
jnz   short loc_43BE8D

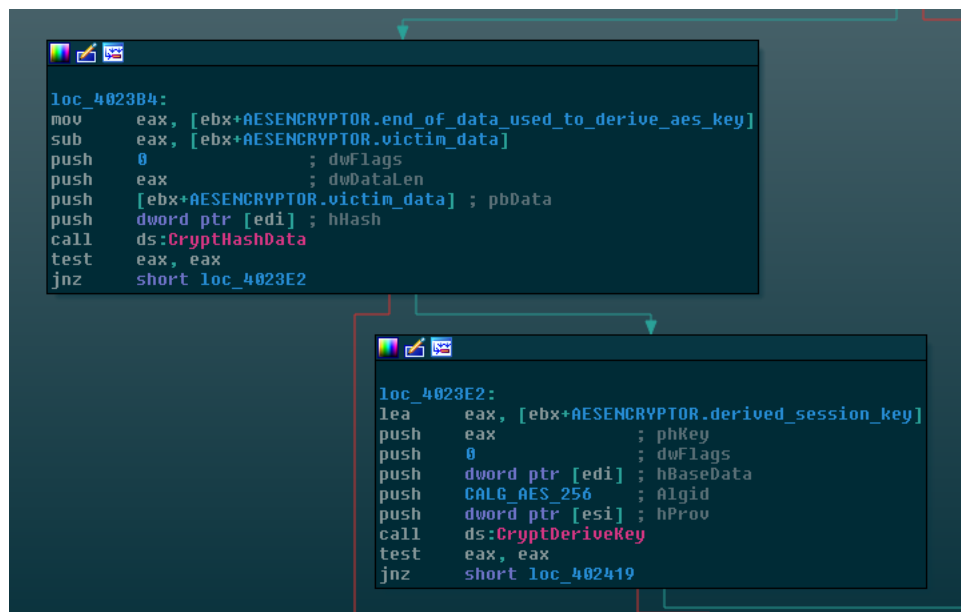
push   edi        ; phKey
push   8000011h  ; dwFlags
push   AT_KEYEXCHANGE ; AlgId
push   dword ptr [esi] ; hProv
call   ds:CryptGenKey
test   eax, eax
jnz   short loc_43BE8D

```

Figure 6: The malware attempts to access a user key in the "pippo container",

and generates a new one if it is not there.

Another curiosity is the malware's use of a custom-named key container named `pippo container`. Other ransomware pretty much universally set a `null` value for the `szContainer` parameter, which defaults to a new key container — then performs all cryptographic operations with the context still open. It's a piece of boilerplate code, a standard ransomware building block that is known to work and cybercriminals therefore copy, paste and forget; the fact that the authors did not use it outright is evidence for some amount of unhealthy curiosity about the windows cryptographic API and its capabilities.



```
loc_4023B4:
mov     eax, [ebx+AESENCRYPTOR.end_of_data_used_to_derive_aes_key]
sub     eax, [ebx+AESENCRYPTOR.victim_data]
push   0             ; dwFlags
push   eax          ; dwDataLen
push   [ebx+AESENCRYPTOR.victim_data] ; pbData
push   dword ptr [edi] ; hHash
call   ds:CryptHashData
test   eax, eax
jnz    short loc_4023E2

loc_4023E2:
lea    eax, [ebx+AESENCRYPTOR.derived_session_key]
push  eax          ; phKey
push  0             ; dwFlags
push  dword ptr [edi] ; hBaseData
push  CALG_AES_256  ; Algid
push  dword ptr [esi] ; hProv
call  ds:CryptDeriveKey
test  eax, eax
jnz   short loc_402419
```

Figure 7: An AES key is derived from a

hash value.

One final deviation from the “classic” ransomware formula is the use of `CryptDeriveKey` on a hashed value in order to derive an AES key (instead of the traditional `CryptGenKey`). If the hash values were derived deterministically then the resulting key could be reconstructed, but some element of randomness in the hash (or any other number of possible separate tweaks on the client or server side) could render this approach viable. Still, we can't repeat enough that when it comes to cryptography, reinventing the wheel is not advised.

## Conclusion

While the attack is still under investigation, the recent Pay2Key ransomware attacks indicate a new threat actor is joining the trend of targeted ransomware attacks – presenting well designed operation to maximize damage and minimize exposure.

The attack was observed targeting the Israeli private sector so far, but looking at the presented tactics, techniques, and procedures we see a potent actor who has no technical reason to limit his targets list to Israel. The incidents are still under investigation, and we will update this blogpost with new findings if any new findings come to light.

Check Point's [Anti-Ransomware](#) solution defends organizations against the most sophisticated ransomware attacks, and safely recovers encrypted data, ensuring business continuity and productivity. Anti-Ransomware is offered as part of Check Point's comprehensive [endpoint security suite](#), SandBlast Agent, to deliver real-time threat prevention to your organization's endpoints.

## Acknowledgments

We would like to thank the researchers from Clarity Research for their assistance and collaboration during this research.

## Check Point Protections

### Threat Cloud Protections:

- Ransomware.Win32.Pay2Key.TC.a
- Ransomware.Win32.Pay2Key.TC.b

### Threat Emulation Protections:

Trojan.Wins.Cobalt.F

## Appendix



---

Indicators of Compromise

Hashes

<b>SHA256</b>	<b>MD5</b>	<b>SHA1</b>
5BAE961FEC67565FB88C8BCD3841B7090566D8FC12CCB70436B5269456E55C00	F3076ADD8669D1C33CD78B6879E694DE	C3FA7E
EA7ED9BB14A7BDA590CF3FF81C8C37703A028C4FDB4599B6A283D68FDCB2613F	4E615861B6D7D778FDC1AC2A61148FE9	EAFFD
D2B612729D0C106CB5B0434E3D5DE1A5DC9D065D276D51A3FB25A08F39E18467	7DB5DD6F2231DA6EB07D907312B1ABE9	A048C2

**Classes**

---

List of Classes retrieved from [Cutter](#) based on RTTI information in the binary:

#### Base classes:

Cobalt::Common::IDecryptor  
Cobalt::Common::IDirectory  
Cobalt::Common::IEncryptor  
Cobalt::Common::IException  
Cobalt::Common::IExceptionFactory  
Cobalt::Common::IFile  
Cobalt::Common::IFileMatcher

Cobalt::DataAccess::IFileReader  
Cobalt::DataAccess::IFileWriter  
Cobalt::DataAccess::ISystemFileManager  
Cobalt::DataAccess::IWindowsDrive

Cobalt::Peripheral::IPeripheralConnection

#### Classes:

Cobalt::Common::MyselfKiller  
Cobalt::Common::CommonExceptionFactory, Base Classes: : Cobalt::Common::IExceptionFactory

Cobalt::Communication::AbortMessage, Base Classes: : Cobalt::Communication::ISerializableMessage, Cobalt::Communication::IMessage  
Cobalt::Communication::AcknowledgeMessage, Base Classes: : Cobalt::Communication::ISerializableMessage,  
Cobalt::Communication::IMessage  
Cobalt::Communication::ClientInformation, Base Classes: : Cobalt::Communication::ISerializableMessage,  
Cobalt::Communication::IMessage  
Cobalt::Communication::ConfigurationMessage, Base Classes: : Cobalt::Communication::ISerializableMessage,  
Cobalt::Communication::IMessage  
Cobalt::Communication::ExceptionMessage, Base Classes: : Cobalt::Communication::ISerializableMessage,  
Cobalt::Communication::IMessage, Cobalt::Common::IException  
Cobalt::Communication::ExceptionMessage\_1, Base Classes: : Cobalt::Communication::ISerializableMessage,  
Cobalt::Communication::IMessage, Cobalt::Common::IException  
Cobalt::Communication::GetClientInformation, Base Classes: : Cobalt::Communication::ISerializableMessage,  
Cobalt::Communication::IMessage  
Cobalt::Communication::GetIdentificationMessage, Base Classes: : Cobalt::Communication::ISerializableMessage,  
Cobalt::Communication::IMessage  
Cobalt::Communication::IMessage  
Cobalt::Communication::IMessageExtractor  
Cobalt::Communication::IMessageFactory  
Cobalt::Communication::IMessagePackager  
Cobalt::Communication::ISerializableMessage  
Cobalt::Communication::IdentificationMessage, Base Classes: : Cobalt::Communication::ISerializableMessage,  
Cobalt::Communication::IMessage  
Cobalt::Communication::JobFinishedMessage, Base Classes: : Cobalt::Communication::ISerializableMessage,  
Cobalt::Communication::IMessage  
Cobalt::Communication::MessageExtractor, Base Classes: : Cobalt::Communication::IMessageExtractor  
Cobalt::Communication::MessageFactory, Base Classes: : Cobalt::Communication::IMessageFactory  
Cobalt::Communication::MessagePackager, Base Classes: : Cobalt::Communication::IMessagePackager  
Cobalt::Communication::PublicKeyMessage, Base Classes: : Cobalt::Communication::ISerializableMessage,  
Cobalt::Communication::IMessage  
Cobalt::Communication::SessionKey, Base Classes: : Cobalt::Communication::ISerializableMessage, Cobalt::Communication::IMessage

Cobalt::DataAccess::BasicFileReader, Base Classes: : Cobalt::DataAccess::IFileReader  
Cobalt::DataAccess::BasicFileWriter, Base Classes: : Cobalt::DataAccess::IFileWriter  
Cobalt::DataAccess::WindowsDirectory, Base Classes: : Cobalt::Common::IDirectory  
Cobalt::DataAccess::WindowsDriveHandler, Base Classes: : Cobalt::DataAccess::IWindowsDrive  
Cobalt::DataAccess::WindowsFile, Base Classes: : Cobalt::Common::IFile  
Cobalt::DataAccess::WindowsFileManager, Base Classes: : Cobalt::DataAccess::ISystemFileManager

Cobalt::DataProcessing::AESEncryptor, Base Classes: : Cobalt::Common::IEncryptor  
Cobalt::DataProcessing::ExtensionFileMatcher, Base Classes: : Cobalt::Common::IFileMatcher  
Cobalt::DataProcessing::RSAClient, Base Classes: : Cobalt::Common::IEncryptor  
Cobalt::DataProcessing::RSAServer, Base Classes: : Cobalt::Common::IDecryptor

Cobalt::Peripheral::TCPClient, Base Classes: : Cobalt::Peripheral::IPeripheralConnection,  
std::enable\_shared\_from\_this\_class\_Cobalt::Peripheral::TCPClient\_

## Suspected file paths

---

C:\Windows\IME\en-GB\client\Cobalt.Client.exe  
C:\Windows\IME\en-GB\mgr\ConnectPC.exe  
C:\Windows\IME\en-GB\mgr\binPS\PExec.exe  
C:\Windows\Temp\[organization-name]tmp\Cobalt.Client.exe

## Strings Written to the Log

---

Deleting Myself  
Restarting  
Application Closed  
wrong config file  
no config file found  
Prevent ShutDown  
end of main procedure  
message is encrypted  
message is not encrypted  
Cannot initialize RSA encryptor  
Cannot initialize RSA decryptor  
Error: receiving wrong response  
receiving server public key  
Error: receiving wrong response  
receiving session\_key  
Error: receiving Configuration Message  
Start Encrypting Engine  
GetClientInformation message received  
Sending Identification message again  
Receive Abort Message  
Connection Restarted  
Sending Identification Message  
Error in getting ipaddress and macaddress  
Sending public key  
Sending public key finished  
send\_session\_key  
Wait for threads to finish their encrypting job  
End of encrypting  
Change Computer Background  
Send Job Finished Message  
We are not connected to serverr, trying 3 second later  
Send Message in Another Thread  
Send message again time out reached  
Connect Again to Server  
Sending Message Process Finished  
copy file to desktop background path  
Receive Data  
Failed To Get Data...  
Start Searching Details of Drive  
Its in Black Path List