BlackMatter Ransomware Technical Analysis and Tools from Nozomi Networks Labs

By Nozomi Networks

Published: 2025-03-27 · Archived: 2026-04-06 00:39:28 UTC

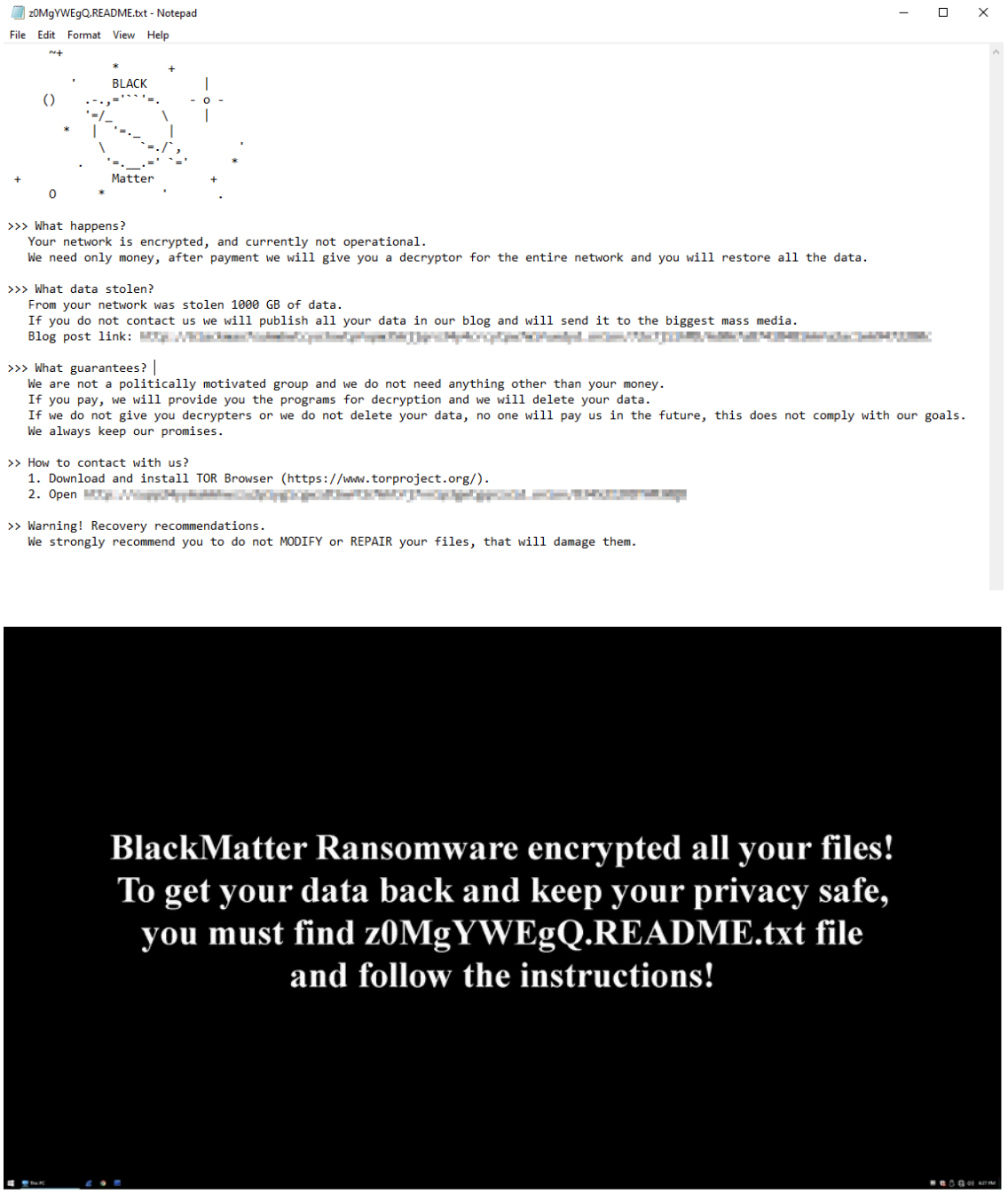
Over the last weekend, Iowa-based NEW Cooperative Inc. was the latest victim of the ransomware group BlackMatter. According to the company, which operates as a farmers' cooperative, the incident has been actively handled, but at the time of this writing the full impact of the attack is not clear.

In the media inquiries section of its website, BlackMatter explicitly lists a series of critical infrastructure targets that should not be targeted by its malicious operations. An organization the size of NEW Cooperative could very well be categorized as critical infrastructure. If that's the case, this attack could have significant consequences. Modern supply chains are sometimes found to be vulnerable to sudden disruptions, with the full effects often understood only much later.

In this blog, we describe the process that Nozomi Networks Labs took to analyze the BlackMatter ransomware executable, as well as ways the malware hinders analysis, and how we were able to overcome them. We provide some scripts that can help other researchers extract key information from other instances of this ransomware that surface in the wild.

Main Functionality

The ransomware encrypts victims' files with a version of the ChaCha20 and RSA algorithms. RSA is used to ensure that decryption is not possible without the private key stored on the attackers' side. The malware leaves a note in the form of a README file with the steps to follow to decrypt them. In addition, it changes the wallpaper to bring attention to them:



Wallpaper changed by the BlackMatter ransomware executable, drawing attention to a README file with decryption steps.*(Click to enlarge)*

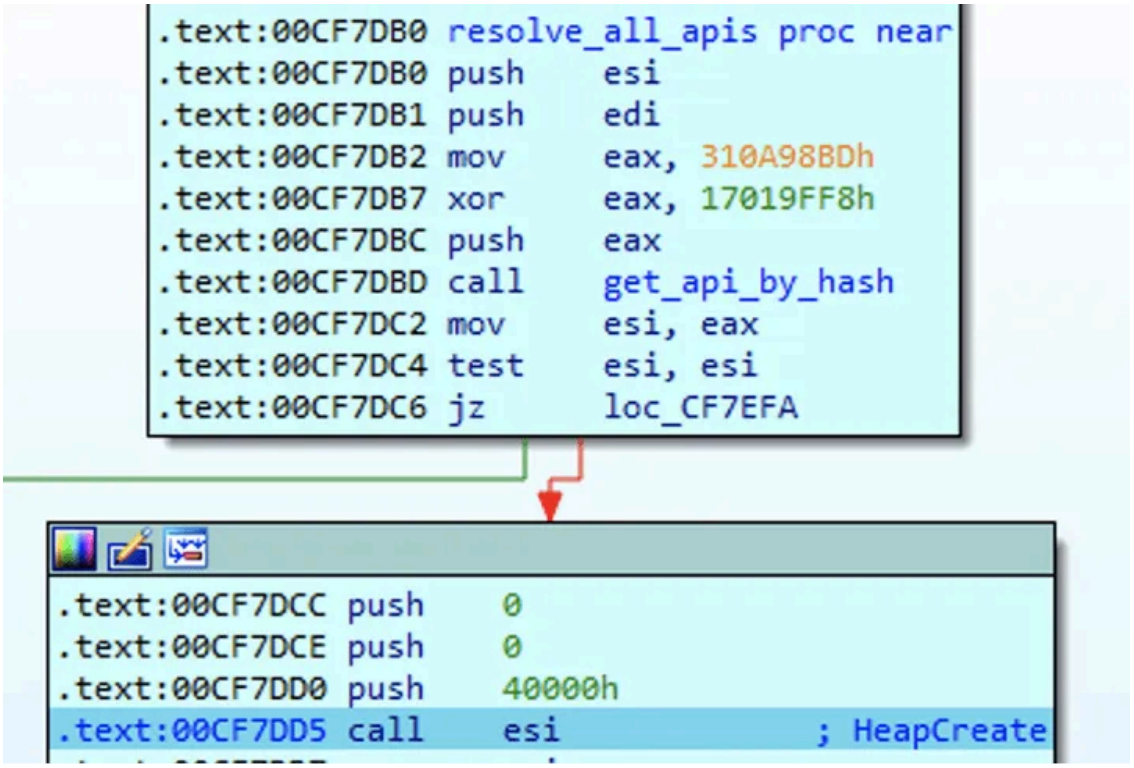
In addition, the malware performs various common ransomware actions such as:

- Deleting shadow copies (local backups) by first listing them using WMI query SELECT * FROM Win32_ShadowCopy
- Deleting files in the recycle bin
- Terminating processes and services specified in the configuration
- Changing the wallpaper to point to the README text file for decryption instructions
- Elevation:Administrator!new:{3E5FC7F9-9A51-4367-9063-A120244FBEC7} is used for UAC (user account control) bypass

- Encrypted files will get a new file extension matching the victim id seen in the README file name prefix and also stored in the registry. This victim id is derived from the MachineGuid registry value.

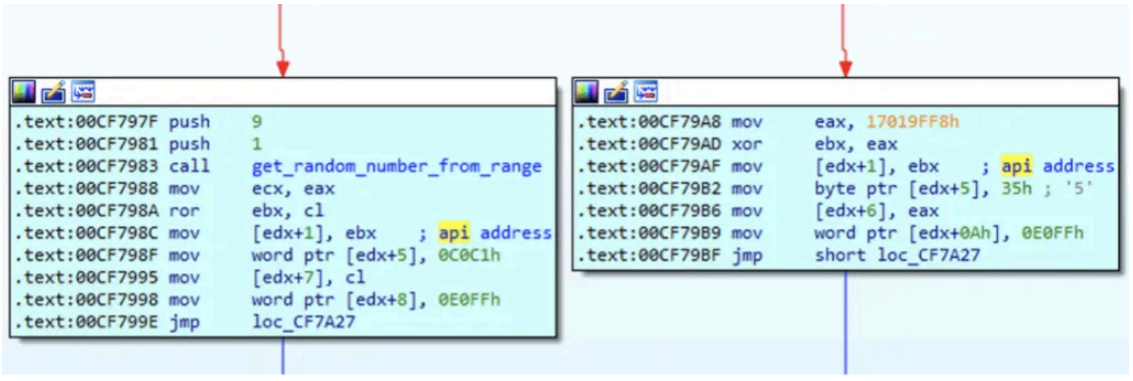
Anti-debugging Techniques

The malware attempts to thwart analysis by hiding which WinAPIs it relies on. To circumvent this, the malware resolves some of the required import functions by their hashes:



Identification of WinAPI function by hashed name

To further complicate analysis, in case of bulk WinAPI address resolution by hashes, the malware uses a unique way of storing the addresses found. Instead of just storing them in a table, for every resolved WinAPI address, it randomly chooses one of five different ways to encode it (rol, ror, xor, xor+rol or xor+ror) and stores the encoded address together with a dynamically built code snippet that will decode it just before the call:



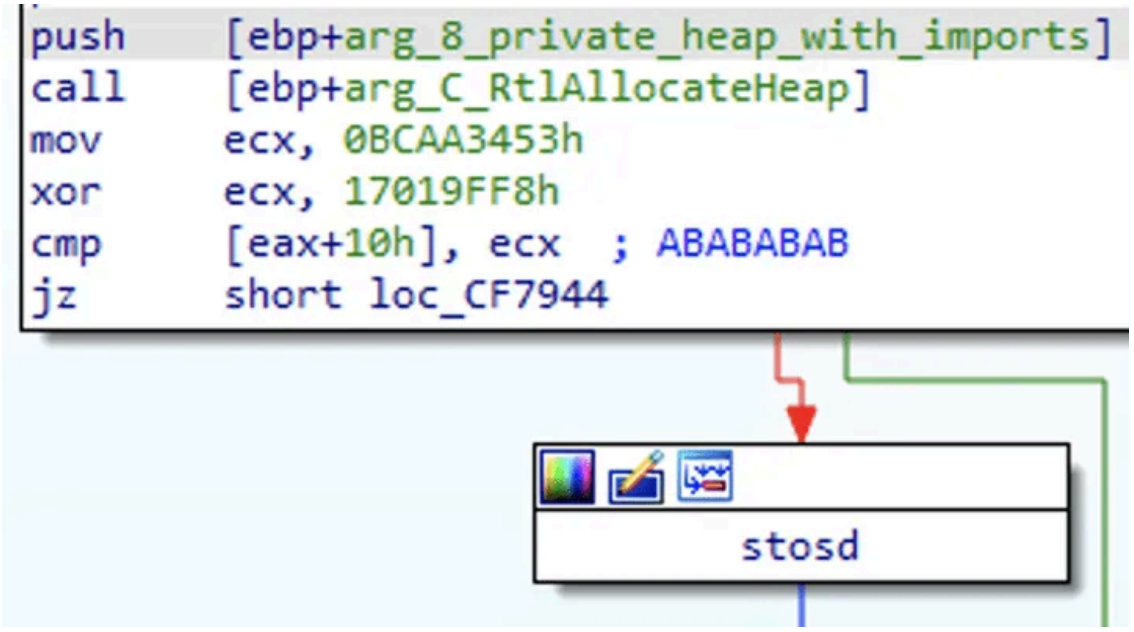
Building code snippets to dynamically decrypt each API address and transfer control to it

Here is one of the result proxy code snippets:

```
B8 71 37 DD C1      mov     eax, 0C1DD3771h
C1 C0 06            rol     eax, 6
FF E0               jmp     eax
```

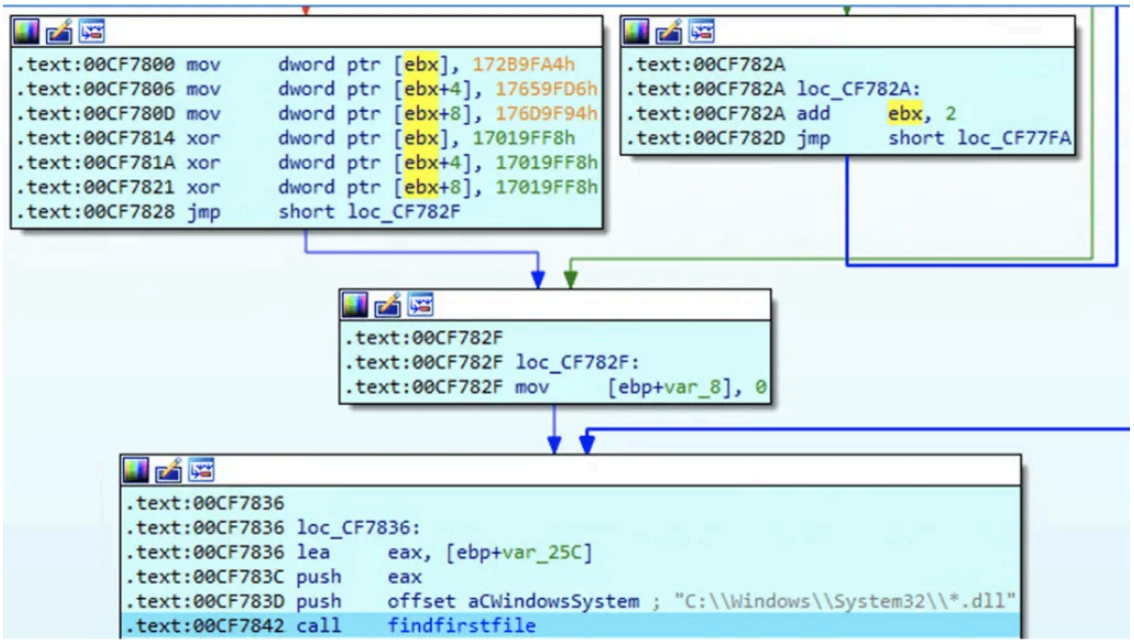
Dynamically built code snippet to call the API

Another anti-debugging trick used by malware is checking the presence of the 0xABABABAB sequence at the end of private heap blocks that it allocates to store these snippets. If the debugger is attached, this sequence will be added and the malware won't store the address of the snippet in its custom import table, which will later result in the debugged sample crashing.

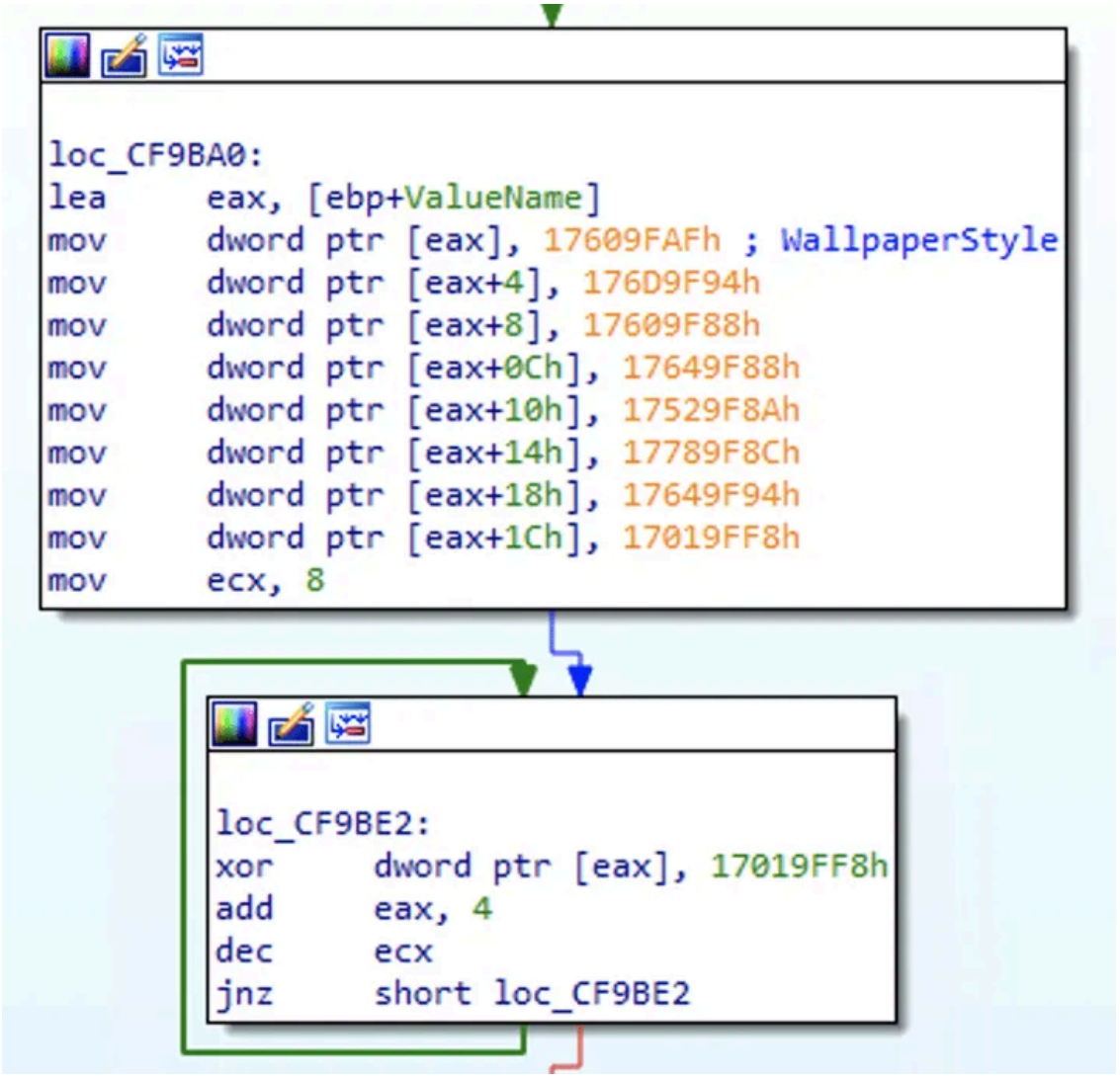


Malware checks for the presence of the 0xABABABAB sequence revealing the debugger

The strings are commonly decrypted on the fly, just before being used:



With the help of IDAPython functionality, it is possible to automatically find and decrypt most of them:

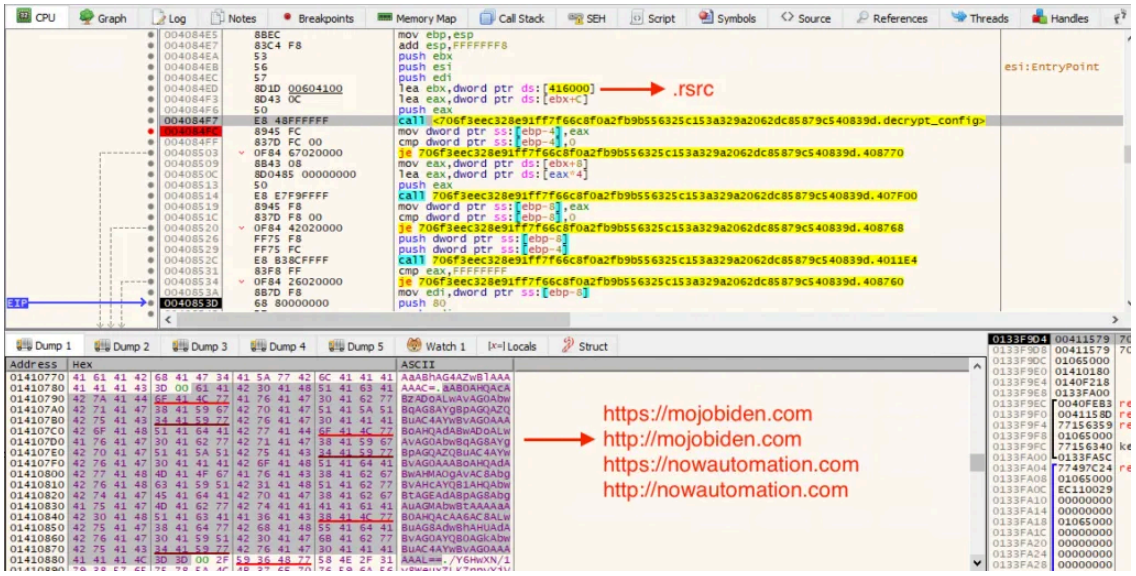


SOFTWARE\Microsoft\Cryptography
MachineGuid
__ProviderArchitecture
ROOT\CIMV2
ID
SELECT * FROM Win32_ShadowCopy
WQL
Win32_ShadowCopy.ID='%s'
Global\%.8x%.8x%.8x%.8x
Times New Roman
.bmp
Control Panel\Desktop
WallPaper
WallpaperStyle
Z:\
dllhost.exe
Elevation:Administrator!new:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}
%s.README.txt
Control Panel\International
LocaleName
sLanguage
SOFTWARE\Microsoft\Windows NT\CurrentVersion
ProductName
%.8x%.8x%.8x%.8x%
POST
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
%s=%s
%s=%s
%.8x%.8x%.8x%.8x%
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
%u.%u
%u.%u
\\%s\
LDAP://rootDSE
defaultNamingContext
LDAP://CN=Computers,
dNSHostName
\\%s\
ExchangeInstallPath
Program Files
Mailbox

SOFTWARE\%s
hScreen

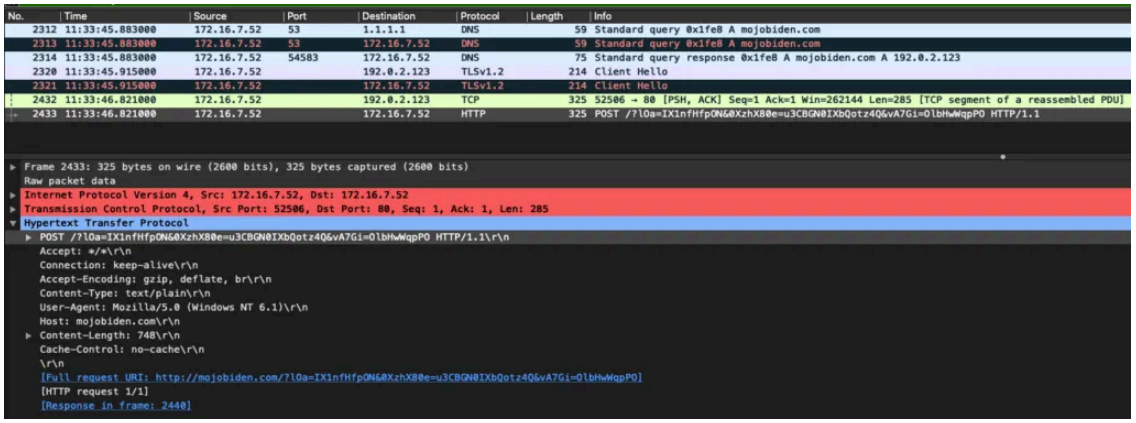
Configuration

The sample's encrypted configuration is stored in the .rsrc section, additionally compressed, and the individual fields are base64-encoded. The decrypted C2 configuration can be seen below. The sample can interact with both plain HTTP and HTTPS endpoints as evidenced by the set of C2.



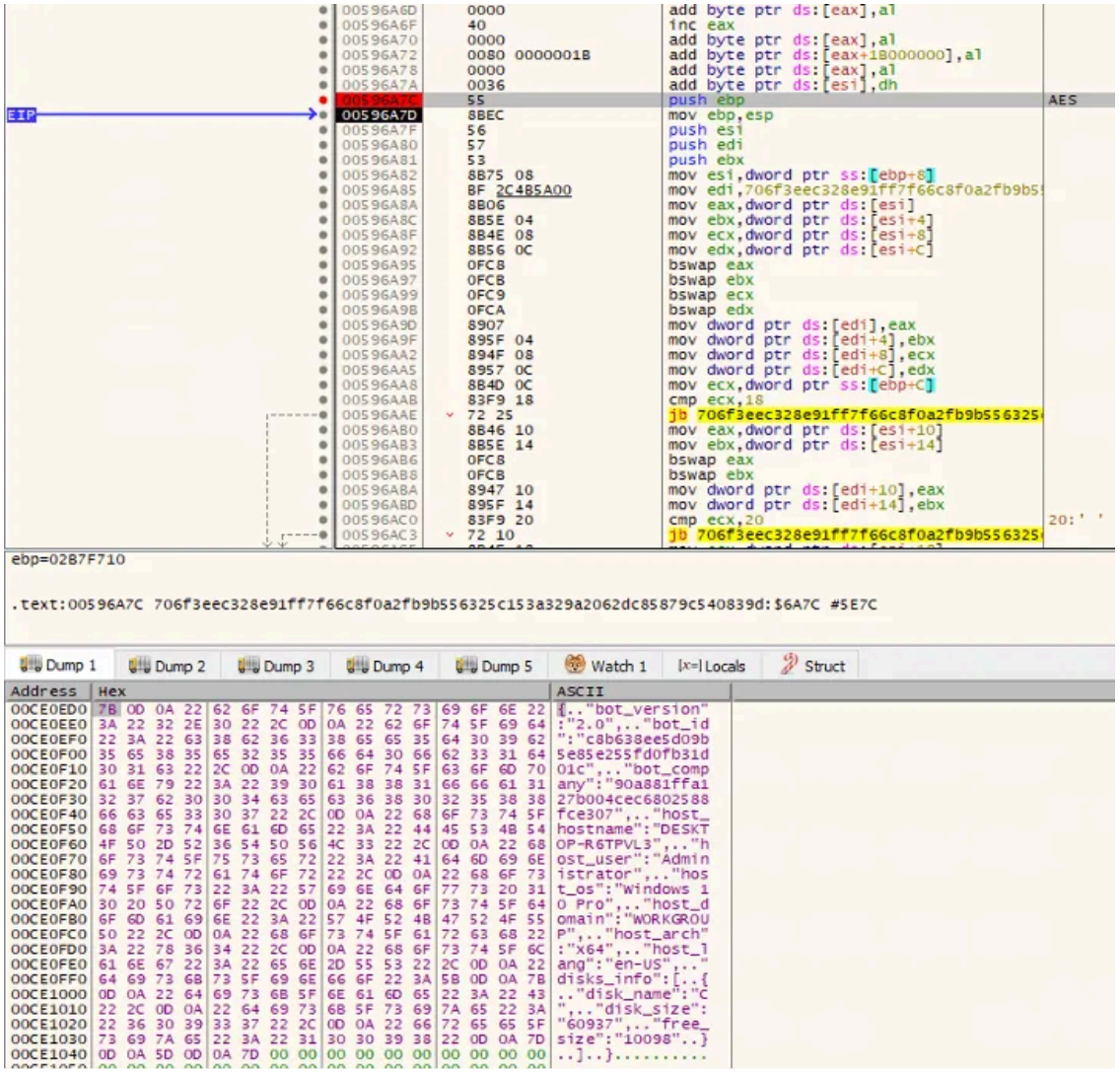
Configuration decryption and base64-encoded C2

Malware generates random HTTP query values when it communicates with these C2:



Network communication with one of the C2

To secure communication, the AES algorithm is used.



Details of the targeted system in plaintext

Here is the extracted configuration:


```
{
 "SHA256_SAMPLE": "706F3EEC328E91FF7F66C8F0A2FB9B556325C153A329A2062DC85879C540839D",
 "RSA_KEY": "232FBA5316E1C9A3F0E603EF0ECB534A1FC1E8BA5F89DBD886D98FBF88EEDDE66CC65E00BBB827CD0262B65C505D95A008
 "COMPANY_VICTIM_ID": "90A881FFA127B004CEC6802588FCE307",
 "AES_KEY": "B59C952C492BD3D1F8F5140AA2855CDE",
 "BOT_MALWARE_VERSION": "2.0",
 "ODD_CRYPT_LARGE_FILES": "false",
 "NEED_MAKE_LOGON": "true",
 "MOUNT_UNITS_AND_CRYPT": "true",
 "CRYPT_NETWORK_RESOURCES_AND_AD": "true",
 "TERMINATE_PROCESSES": "true",
 "STOP_SERVICES_AND_DELETE": "true",
 "CREATE_MUTEX": "true",
 "PREPARE_VICTIM_DATA_AND_SEND": "true",
 "PRINT_RANSOM_NOTE": "true",
```

```
"PROCESS_TO_KILL": [
  { "": "encsvc" },
  { "": "thebat" },
  { "": "mydesktopqos" },
  { "": "xfssvccon" },
  { "": "firefox" },
  { "": "infopath" },
  { "": "winword" },
  { "": "steam" },
  { "": "synctime" },
  { "": "notepad" },
  { "": "ocomm" },
  { "": "onenote" },
  { "": "mspub" },
  { "": "thunderbird" },
  { "": "agntsvc" },
  { "": "sql" },
  { "": "excel" },
  { "": "powerpnt" },
  { "": "outlook" },
  { "": "wordpad" },
  { "": "dbeng50" },
  { "": "isqlplussvc" },
  { "": "sqbcoreservice" },
  { "": "oracle" },
  { "": "ocautoupds" },
  { "": "dbsnmp" },
  { "": "msaccess" },
  { "": "tbirdconfig" },
  { "": "ocssd" },
  { "": "mydesktopservice" },
  { "": "visio" }
],
"SERVICES_TO_KILL": [
  { "": "mepocs" },
  { "": "memtas" },
  { "": "veeam" },
  { "": "svc$" },
  { "": "backup" },
  { "": "sql" },
  { "": "vss" },
  { "": "msexchange" }
],
"C2_URLS": [
  { "": "https://mojobiden[.]com" },
  { "": "http://mojobiden[.]com" },
  { "": "https://nowautomation[.]com" },
```

```
    { "": "http://nowautomation[.]com" }
  ],
  "LOGON_USERS_INFORMATION": [
    { "": "" },
    { "": "" },
    { "": "" },
    { "": "" },
    { "": "" },
    { "": "" }
  ],
  "RANSOM_NOTE": [
    {
      "": "      ~+                                          \r\n              *       +\r\n        '
    }
  ]
}
```


Overall, there are multiple similarities with the DarkSide ransomware family, including the way the victim id is derived from the MachineGuid value, the encryption techniques used, and the way the configuration is structured and protected. More information on the DarkSide executable can be found in our previous blog.

BlackMatter Ransomware Protection and Indicators of Compromise

Nozomi Networks customers using our Threat Intelligence service are already covered against the described threat. In addition, Nozomi Networks Labs is monitoring this situation as it evolves and will extend coverage to customers and keep the community informed of major updates.

For security professionals defending critical infrastructure operations, general recommendations for cyber resiliency against ransomware is found in our latest OT/IoT Security Report.

For security researchers, the descriptions provided in this blog of how BlackMatter evades analysis, and how to extract key information from the code should be useful as the malware evolves.

The indicators of compromise (IOC) that we learned from this analysis, as well as the scripts we used in the analysis are found below.

List of IOCs

mojobiden.com nowautomation.com 706f3eec328e91ff7f66c8f0a2fb9b556325c153a329a2062dc85879c540839d

```
 // Created by Nozomi Networks Labs

import "pe"

rule blackmatter_ransomware : blackmatter ransomware {
    meta:
```

```
        date = "2021-09-20"
        name = "BlackMatter - RANSOMWARE"
        author = "Nozomi Networks Labs"
        description = "Generic detection for BlackMatter ransomware"
        actor = "BlackMatter"
        x_threat_name = "BlackMatter ransomware"
        x_mitre_technique = "T1486"
        hash1 = "706f3eec328e91ff7f66c8f0a2fb9b556325c153a329a2062dc85879c540839d"
        hash2 = "9cf9441554ac727f9d191ad9de1dc101867ffe5264699cafcf2734a4b89d5d6a"
        hash3 = "b0e929e35c47a60f65e4420389cad46190c26e8cfaabe922efd73747b682776a"
        hash4 = "2cdb5edf3039863c30818ca34d9240cb0068ad33128895500721bcdca70c78fd"
        hash5 = "f7b3da61cb6a37569270554776dbbd1406d7203718c0419c922aa393c07e9884"
        hash6 = "8f1b0affffb2f2f58b477515d1ce54f4daa40a761d828041603d5536c2d53539"
        hash7 = "e4a2260bcba8059207fdcc2d59841a8c4ddbe39b6b835feef671bceb95cd232d"
        nn_ts = "1632088800.0"
        nn_sig = "f7c69f3b527ffb3f0c2aa613e902d8d4f0e39966048bb6cfa57556115fa18ed9"
        nn_id = "92f90d15-9392-4076-96b5-1e42ac9874c5"

    condition:
        uint16(0) == 0x5a4d and
        uint32(uint32(0x3c)) == 0x00004550 and
        filesize < 100KB and
        pe.imphash() == "2e4ae81fc349a1616df79a6f5499743f"
}
```

IDAPython Scripts

Here is a script to restore the custom import table dynamically populated by malware. It defines the new hotkey Z that should be pressed when the cursor is located at the bulk decryption function (in case of this sample, at the RVA 0x78EC).

```
 # Author: Alexey Kleymenov (a member of Nozomi Networks Labs)

import os
import struct
import pefile
import ida_kernwin

PATH_TO_DLLS = 'c:\\windows\\system32\\'
HARDCODED_XOR_KEY = 0x17019FF8

def extract_api_hashes(start):
    '''
    Returns a dictionary where keys are import functions to write data and values are list of hashes.
    The first hash is the DLL name's hash, the rest are WinAPI names' hashes.
    '''
```

```
    decryptor_address = start
    print('Bulk API decryptor address: %x' % decryptor_address)

    api_hashes = {}
    for head in Heads():
        flags = GetFlags(head)
        if isCode(flags):
            prev = prev_head(head)
            prev_2 = prev_head(prev)

            if print_insn_mnem(head) == 'call' and get_operand_value(head, 0) == decryptor_address:
                print('Found the decryptor called: %x' % head)

                if print_insn_mnem(prev) == 'push' and print_insn_mnem(prev_2) == 'push':
                    func_hashes = get_operand_value(prev_2, 0)
                    import_table = get_operand_value(prev, 0)
                    api_hashes[import_table] = []

                    for i in range(0, 0xffff, 4):
                        api_hash = struct.unpack("<I", get_bytes(func_hashes + i, 4))[0]
                        if api_hash == 0xCCCCCCCC:
                            break
                        else:
                            api_hashes[import_table].append(api_hash ^ HARDCODED_XOR_KEY)
                else:
                    print('Non-standard arguments %x' % head)
    return api_hashes

def calculate_checksum(name, value):
    '''Standard ror 0x0D'''
    for symbol in name:
        value = ((value >> 0x0D) | (value << (0x20 - 0x0D))) & 0xFFFFFFFF
        value += ord(symbol) & 0xFFFFFFFF
    return value

def build_mappings(dll_filepath, dll_hashes):
    '''
    Calculates API checksums for the DLLs of interest
    '''
    dll_name = os.path.basename(dll_filepath)
    dll_checksum = calculate_checksum(dll_name.lower() + '\x00', 0)
    result = {}

    if dll_checksum in dll_hashes:
        dll = pefile.PE(dll_filepath, fast_load=True)
        dll.parse_data_directories(directories=[pefile.DIRECTORY_ENTRY['IMAGE_DIRECTORY_ENTRY_EXPORT']])
```

```
        if hasattr(dll, 'DIRECTORY_ENTRY_EXPORT'):
            dll_name = dll_name.replace('.', '_')
            result[dll_checksum] = {'dll_name': dll_name}
            export_directory = dll.DIRECTORY_ENTRY_EXPORT

            for symbol in export_directory.symbols:
                if symbol.name is not None:
                    api_name = symbol.name.decode('latin-1')
                    api_checksum = calculate_checksum(api_name + '\x00', dll_checksum)
                    result[api_checksum] = {'dll_name': dll_name, 'api_name': api_name}
    return result

def parse_dlls(path_to_dlls, dll_hashes):
    '''
    Walks all files in the given path and builds export hash mappings
    '''
    list_dlls = os.listdir(path_to_dlls)
    mappings = {}

    for dll_filename in list_dlls:
        full_path = os.path.join(path_to_dlls, dll_filename)
        mappings.update(build_mappings(full_path, dll_hashes))

    return mappings

def decrypt_all():
    '''
    Should be run with the cursor at the bulk decryption function
    '''
    start = get_screen_ea()
    api_hashes = extract_api_hashes(start)
    dll_hashes = []

    for _, hashes in api_hashes.items():
        dll_hashes.append(hashes[0])

    dll_mappings = parse_dlls(PATH_TO_DLLS, dll_hashes)

    for import_table, hashes in api_hashes.items():
        dll_hash = hashes[0]
        api_hashes = hashes[1:]

        if dll_hash in dll_mappings:
            print('Found DLL hash %x = %s' % (dll_hash, dll_mappings[dll_hash]['dll_name']))

            for i, api_hash in enumerate(api_hashes):
                if api_hash in dll_mappings:
```

```
                    addr = import_table + (i + 1) * 4
                    print('Found API hash for %x = %s (%s)' % (
                        addr,
                        dll_mappings[api_hash]['api_name'],
                        dll_mappings[api_hash]['dll_name']
                    ))
                    set_name(addr, dll_mappings[api_hash]['api_name'])
                else:
                    print('API hash %x not found' % api_hash)
        else:
            print('DLL hash %x not found' % dll_hash)

ida_kernwin.add_hotkey("z", decrypt_all)


# Additional: Search & Decrypt Encrypted Strings

# Author: Alexey Kleymenov (a member of Nozomi Networks Labs)

import struct
import ida_kernwin

HARDCODED_XOR_KEY = 0x17019FF8

def is_utf16_heur(string):
    counter = 0
    for val in string:
        if val == 0:
            counter += 1
    if counter / float(len(string)) > 0.4:
        return True
    return False

def decrypt_string(start_addr):
    addr = start_addr
    result = b""

    for i in range(0xFFFF):
        instr = print_insn_mnem(addr)
        if instr != 'mov' or 'dword ptr' not in GetDisasm(addr):
            break

        value = get_operand_value(addr, 1)
        decoded_value = value ^ HARDCODED_XOR_KEY
        result += struct.pack("<I", decoded_value)
        addr = next_head(addr)
```

```
    result_orig = result

    if is_utf16_heur(result):
        result = result.decode('utf-16le')
    else:
        result = result.decode('latin-1')

    if all(ord(c) < 128 for c in result):
        result = result.rstrip('\x00')
    else:
        result = 'hex: ' + result_orig.hex()

    print('%x - %s' % (start_addr, result))
    set_cmt(start_addr, result, 0)

def decrypt_string_manual():
    start_addr = get_screen_ea()
    decrypt_string(start_addr)

def search_for_encrypted_strings():
    for head in Heads():
        flags = GetFlags(head)
        if isCode(flags):
            if print_insn_mnem(head) == 'xor' and 'dword ptr' in GetDisasm(head) and get_operand_value(head, 1)
                next = next_head(head)
                if print_insn_mnem(next) == 'add' and get_operand_value(next, 1) == 4:
                    prev = prev_head(head)
                    if 'mov     ecx' in GetDisasm(prev):
                        num = get_operand_value(prev, 1)
                        for i in range(num):
                            prev = prev_head(prev)
                        # print('Found the encryption string candidate: %x' % prev)
                        decrypt_string(prev)

ida_kernwin.add_hotkey(",", decrypt_string_manual)
search_for_encrypted_strings()
```

References:

1. https://github.com/advanced-threat-research/DarkSide-Config-Extract

---