Colonial Pipeline Ransomware Attack: Revealing How DarkSide Works

nozominetworks.com/blog/colonial-pipeline-ransomware-attack-revealing-how-darkside-works/

By May 19, 2021

Throughout the last two weeks, the entire cybersecurity community has been riveted by the Colonial Pipeline ransomware attack. It is one of the most notable attacks on critical infrastructure of the past few years and has directly and indirectly impacted multiple industries in the U.S economy. Thankfully, operations are up and running after an approximately week-long outage and reported payment of a $5 million ransom.[1]

DarkSide, the Ransomware as a Service (RaaS) deployed against Colonial Pipeline, is a good example of similar malware attacking organizations around the globe. Carefully prepared and deployed, it uses a combination of techniques to successfully extort its victims.

Nozomi Networks Labs has studied the internals of the DarkSide executable and today we're sharing our findings to reveal the techniques used by its machine code in three areas: the selection of victims and files, ensuring anonymity and anti-detection, and preventing data restoration. We also provide IoCs and a decryption script to help you detect DarkSide.

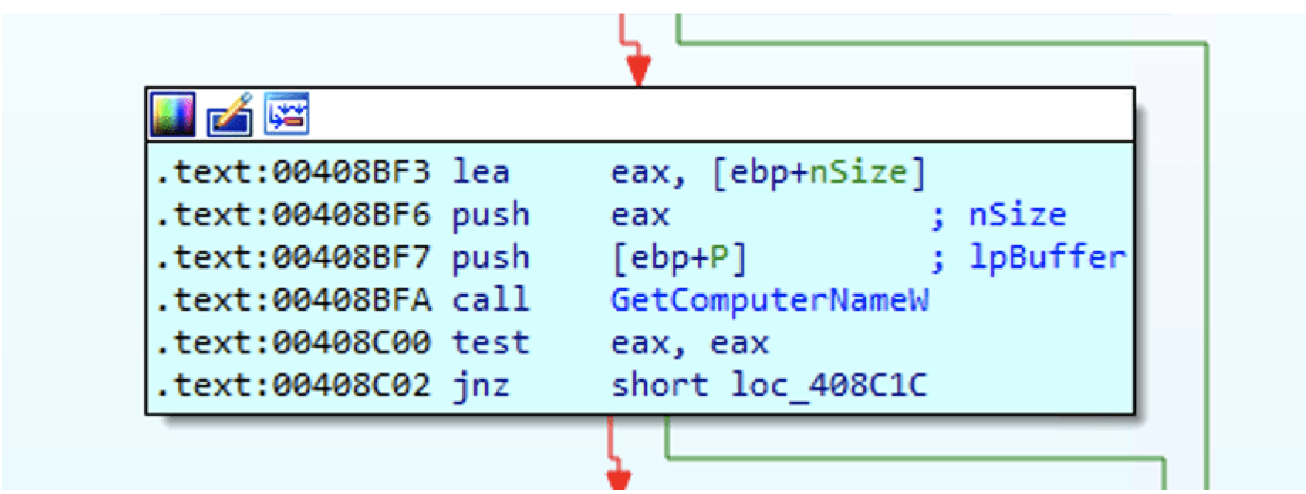
It's important to remember that the sum of Darkside's code translates to devastating consequences in the physical world. We encourage you to understand DarkSide's techniques to help you assess both your own defenses and your incident response capabilities.

Nozomi Networks Labs’ technical analysis of the DarkSide ransomware’s main executable reveals the techniques it uses – helping you assess your own defenses.

DarkSide Ransomware: Technical Analysis

Victim Validation

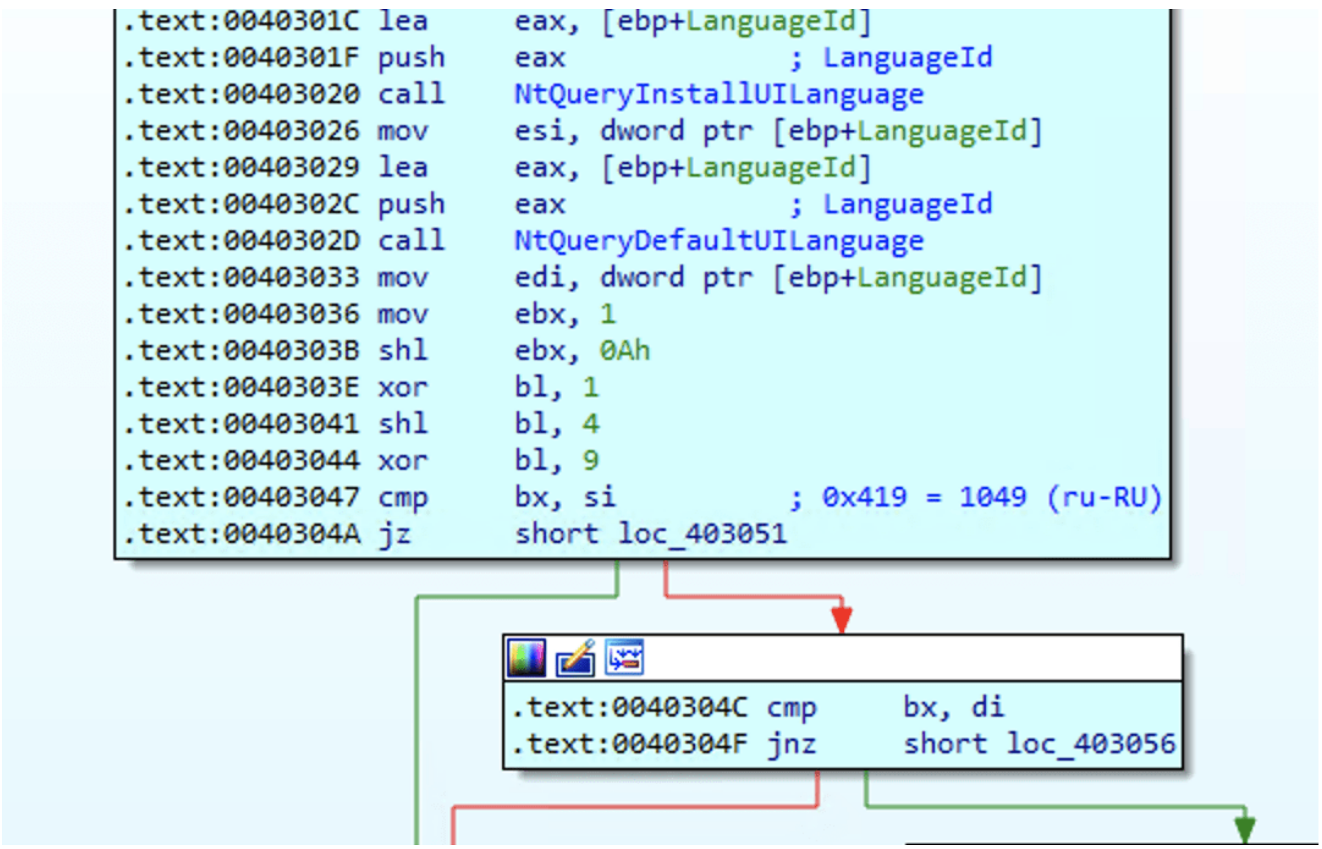
The malware first collects basic information about its victim’s computer systems to learn the details of the technical environment.



The malware obtains the affected computer’s name.

```
aOsLangSUsernam:                        ; DATA XREF: collect_system_info+12↑o
                text "UTF-16LE", '"os":{',0Dh,0Ah
                text "UTF-16LE", '"lang":"%s",',0Dh,0Ah
                text "UTF-16LE", '"username":"%s",',0Dh,0Ah
                text "UTF-16LE", '"hostname":"%s",',0Dh,0Ah
                text "UTF-16LE", '"domain":"%s",',0Dh,0Ah
                text "UTF-16LE", '"os_type":"windows",',0Dh,0Ah
                text "UTF-16LE", '"os_version":"%s",',0Dh,0Ah
                text "UTF-16LE", '"os_arch":"%s",',0Dh,0Ah
                text "UTF-16LE", '"disks":"%s",',0Dh,0Ah
                text "UTF-16LE", '"id":"%s"',0Dh,0Ah
                text "UTF-16LE", '}',0
```
DarkSide collects the victim’s basic system information.

In addition, it skips victims from certain geographical regions by checking the language used by their systems. (Notably, DarkSide does not attack systems that use Russian or other Eastern European languages.[2])



The ransomware checks if the system language is the one used in CIS countries.

Selection of Files for Encryption

Next, DarkSide determines what files to encrypt. If malware attempts to encrypt all the files available on the system, it quickly makes the system unusable – and leaves the victim without information on how contact the attackers. In addition, it takes significantly more time to do the encryption than is needed for the purposes of executing the attack. DarkSide is particularly selective about what files it encrypts, selecting them mainly by examining their file directories, file names and file extensions.

```
$recycle.bin config.msi $windows.~bt $windows.~ws windows appdata application data boot google mozilla
program files program files (x86) programdata system volume information tor browser windows.old intel m
socache perflogs x64dbg public all users default
autorun.inf boot.ini bootfont.bin bootsect.bak desktop.ini iconcache.db ntldr ntuser.dat ntuser.dat.log
 ntuser.ini thumbs.db
386 adv ani bat bin cab cmd com cpl cur deskthemepack diagcab diagcfg diagpkg dll drv exe hlp icl icns
ico ics idx ldf lnk mod mpa msc msp msstyles msu nls nomedia ocx prf ps1 rom rtp scr shs spl sys theme
themepack wpx lock key hta msi pdb
```

The list of directories, file names and file extensions skipped during the encryption.

Anonymity

To remain anonymous and prevent prompt shutdown, websites for contacting ransomware threat actors are hosted in the Tor network.

```
What guarantees?
---------------------------------------------
We value our reputation. If we do not do our work and liabilities, nobody will pay us. This is not in our interests.
All our decryption software is perfectly tested and will decrypt your data. We will also provide support in case of problems.
We guarantee to decrypt one file for free. Go to the site and contact us.

How to get access on website?
---------------------------------------------
Using a TOR browser:
1) Download and install TOR browser from this site: https://torproject.org/
2) Open our website: http://dark24zz[redacted].onion/W57MRI9C7YZJUZEABBBYRQLSUTG22JZ9MAH6
```

A section of DarkSide's instructions describing how to access the Tor-based website.

Anti-Detection Techniques

To stay under the radar until the victim's systems are impacted, DarkSide incorporates various commonly used techniques.

Self-Encryption

Most of the Darkside's critical strings are encrypted to avoid triggering detection.

```
.0040B0FB: 33C0              xor     eax,eax
.0040B0FD: 029A51014100     1add     bl,[edx][000410151]
.0040B103: 8A8251014100      mov     al,[edx][000410151]
.0040B109: 8AAB50014100      mov     ch,[ebx][000410150]
.0040B10F: 888350014100      mov     [ebx][000410150],al
.0040B115: 88AA51014100      mov     [edx][000410151],ch
.0040B11B: 02C5              add     al,ch
.0040B11D: 47                inc     edi
.0040B11E: 8A8050014100      mov     al,[eax][000410150]
.0040B124: FEC2              inc     dl
.0040B126: 3007              xor     [edi],al
.0040B128: FEC9              dec     cl
.0040B12A: 75D1              jnz    .00040B0FD --↑1
```

The XOR-based decryption algorithm.

For the same reason, the malware's main configuration is also encrypted. It is compressed with aPLib, with individual configuration values encoded with a Base64 algorithm.

```
.text:00402C5D lea     eax, [ebx+24h]
.text:00402C60 push    eax
.text:00402C61 call    decrypt_block
.text:00402C66 mov     [ebp+decrypted_config_block], eax
.text:00402C69 cmp     [ebp+decrypted_config_block], 0
.text:00402C6D jz      loc_403007

.text:00402C73 mov     eax, [ebx+20h]
.text:00402C76 lea     eax, ds:0[eax*4]
.text:00402C7D push    eax              ; Size
.text:00402C7E push    8                ; Flags
.text:00402C80 push    data             ; HeapHandle
.text:00402C86 call    RtlAllocateHeap
.text:00402C8C mov     [ebp+P_config_block_buffer], eax
.text:00402C8F cmp     [ebp+P_config_block_buffer], 0
.text:00402C93 jz      loc_402FF6

.text:00402C99 push    [ebp+P_config_block_buffer]
.text:00402C9C push    [ebp+decrypted_config_block]
.text:00402C9F call    aplib_decompress
.text:00402CA4 cmp     eax, 0FFFFFFFFh
.text:00402CA7 jz      loc_402FE5
```

Decryption and decompression of the configuration block.


```
debug051:005C95E6 aJabyaguaywb5ag db 'JAByAGUAYwB5AGMAbABlAC4AYgBpAG4AAABjAG8AbgBmAGkAZwAuAG0AcwBpAAAAJ'
debug051:005C95E6 db 'AB3AGkAbgBkAG8AdwBzAC4AfgBiAHQAAAAkAHcAaQBuAGQAbwB3AHMALgB+AHcAcw'
debug051:005C95E6 db 'AAAHcAaQBuAGQAbwB3AHMAAABhAHAAcABkAGEAdABhAAAAYQBwAHAAbABpAGMAYQB'
debug051:005C95E6 db '0AGkAbwBuACAAZABhAHQAYQAAAGIAbwBvAHQAAABnAG8AbwBnAGwAZQAAAG0AbwB6'
debug051:005C95E6 db 'AGkAbABsAGEAAABwAHIAbwBnAHIAYQBtACAAZgBpAGwAZQBzAAAAcAByAG8AZwByA'
debug051:005C95E6 db 'GEAbQAgAGYAaQBsAGUAcwAgACgAeAA4ADYAKQAAAHAAcgBvAGcAcgBhAG0AZABhAH'
debug051:005C95E6 db 'QAYQAAAHMAeQBzAHQAZQBtACAAdgBvAGwAdQBtAGUAIABpAG4AZgBvAHIAbQBhAHQ'
debug051:005C95E6 db 'AaQBvAG4AAAB0AG8AcgAgAGIAcgBvAHcAcwBlAHIAAAB3AGkAbgBkAG8AdwBzAC4A'
debug051:005C95E6 db 'bwBsAGQAAABpAG4AdABlAGwAAABtAHMAbwBjAGEAYwBoAGUAAABwAGUAcgBmAGwAb'
debug051:005C95E6 db 'wBnAHMAAAB4ADYANABkAGIAZwAAAHAAdQBiAGwAaQBjAAAAYQBsAGwAIAB1AHMAZQ'
debug051:005C95E6 db 'ByAHMAAABkAGUAZgBhAHUAbAB0AAAAAAB=',0
debug051:005C9893 aYqb1ahqabwbyah db 'YQB1AHQAbwByAHUAbgAuAGkAbgBmAAAAYgBvAG8AdAAuAGkAbgBpAAAAYgBvAG8Ad'
debug051:005C9893 db 'ABmAG8AbgB0AC4AYgBpAG4AAABiAG8AbwB0AHMAZQBjAHQALgBiAGEAawAAAGQAZQ'
debug051:005C9893 db 'BzAGsAdABvAHAALgBpAG4AaQAAAGkAYwBvAG4AYwBhAGMAaABlAC4AZABiAAAAbgB'
debug051:005C9893 db '0AGwAZAByAAAAbgB0AHUAcwBlAHIALgBkAGEAdAAAAG4AdAB1AHMAZQByAC4AZABh'
debug051:005C9893 db 'AHQALgBsAG8AZwAAAG4AdAB1AHMAZQByAC4AaQBuAGkAAAB0AGgAdQBtAGIAcwAuA'
debug051:005C9893 db 'GQAYgAAAAAA',0
```

Base64-encoded configuration values in the malware.

Dynamic API Resolution

WinAPIs are the standard way programs interact with the Windows operating system to access certain functionality, including file and network operations. Therefore, use of these interfaces quickly reveals the actual purpose of the malware to security systems.

To prevent detection, DarkSide does not immediately have all the APIs used available in the import table, as legitimate executables do. Instead, it resolves them dynamically before using them, some by hashed names and some by encrypted names.

```
.text:00401948 push    1E2B04A4h        ; LoadLibrary
.text:0040194D push    3B98045Eh        ; kernel32
.text:00401952 call    resolve_API_by_hash ; LoadLibrary
.text:00401957 mov     LoadLibraryA, eax
.text:0040195C push    288B0588h
.text:00401961 push    3B98045Eh
.text:00401966 call    resolve_API_by_hash ; GetProcAddress
.text:0040196B mov     GetProcAddress, eax
.text:00401970 mov     esi, offset encrypted_api_names
.text:00401975 mov     ebx, 2Eh ; '.'
.text:0040197A mov     edi, offset RtlAllocateHeap
.text:0040197F call    resolve_API_by_name
.text:00401984 mov     ebx, 37h ; '7'
.text:00401989 mov     edi, offset SetFileAttributesW
.text:0040198E call    resolve_API_by_name
.text:00401993 mov     ebx, 15h
.text:00401998 mov     edi, offset LookupAccountSidW
.text:0040199D call    resolve_API_by_name
.text:004019A2 mov     ebx, 8
.text:004019A7 mov     edi, offset GetDC
.text:004019AC call    resolve_API_by_name
.text:004019B1 mov     ebx, 0Dh
```

The dynamic WinAPI resolution used by DarkSide.

Preventing Data Restoration

If system administrators could quickly and easily restore the affected data without paying money to criminals, ransomware attacks would not succeed. The authors of DarkSide incorporate multiple techniques to ensure ransom is paid.

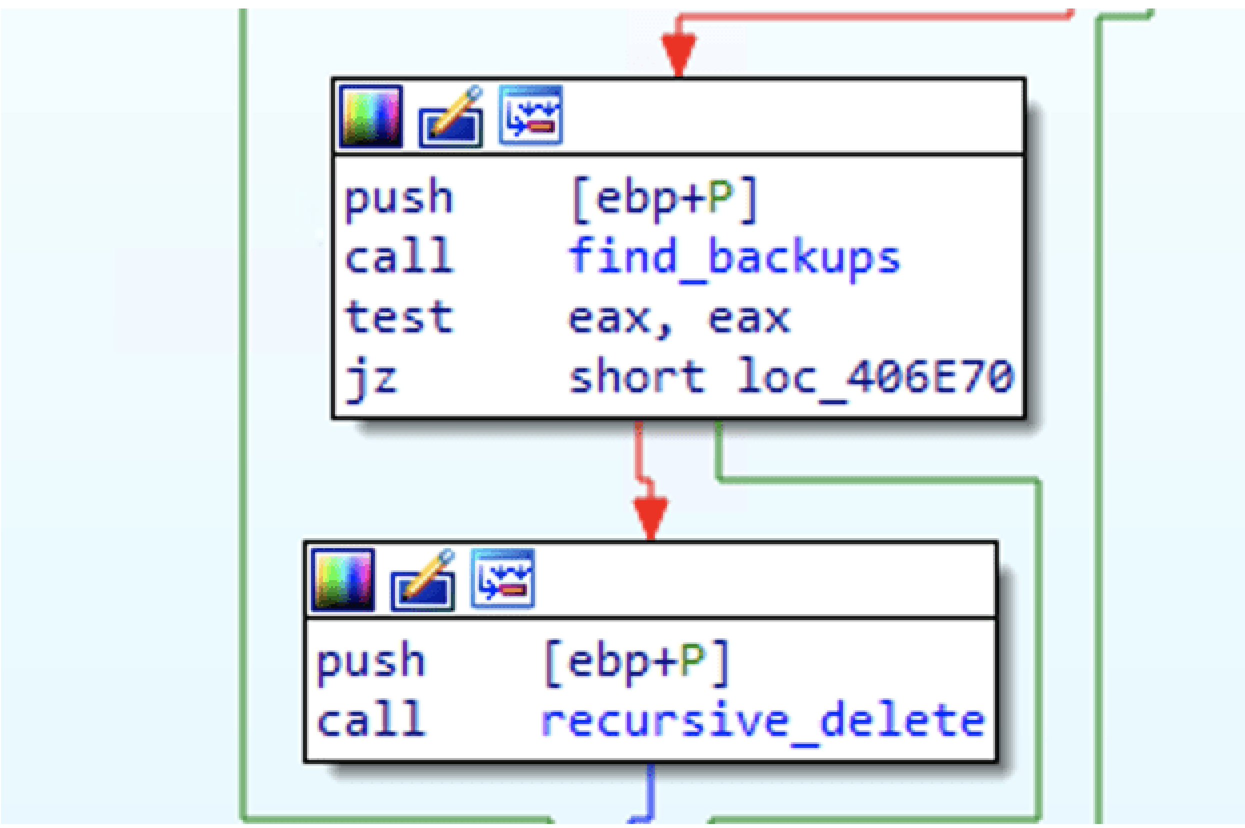
Dealing with Backups

Ransomware makes sure that standard backup solutions are unusable on the targeted machines. Windows has a feature called Shadow Copy aimed at dealing with such situations. It allows the creation of backup copies of computer files so they can be restored when needed. The main limitation of this approach is that the backup files are stored on the same system as the original files. If malware compromises the system, the backup files are readily deleted.

```
loc_4046D9:
push    offset aWql     ; "WQL"
call    decrypt_block
mov     [ebp+P], eax
push    offset aSelectFromWin3 ; "SELECT * FROM Win32_ShadowCopy"
call    decrypt_block
mov     [ebp+var_C], eax
```

The commands used to get a list of Shadow Copy backups.

In addition, the malware can search for backups by name:

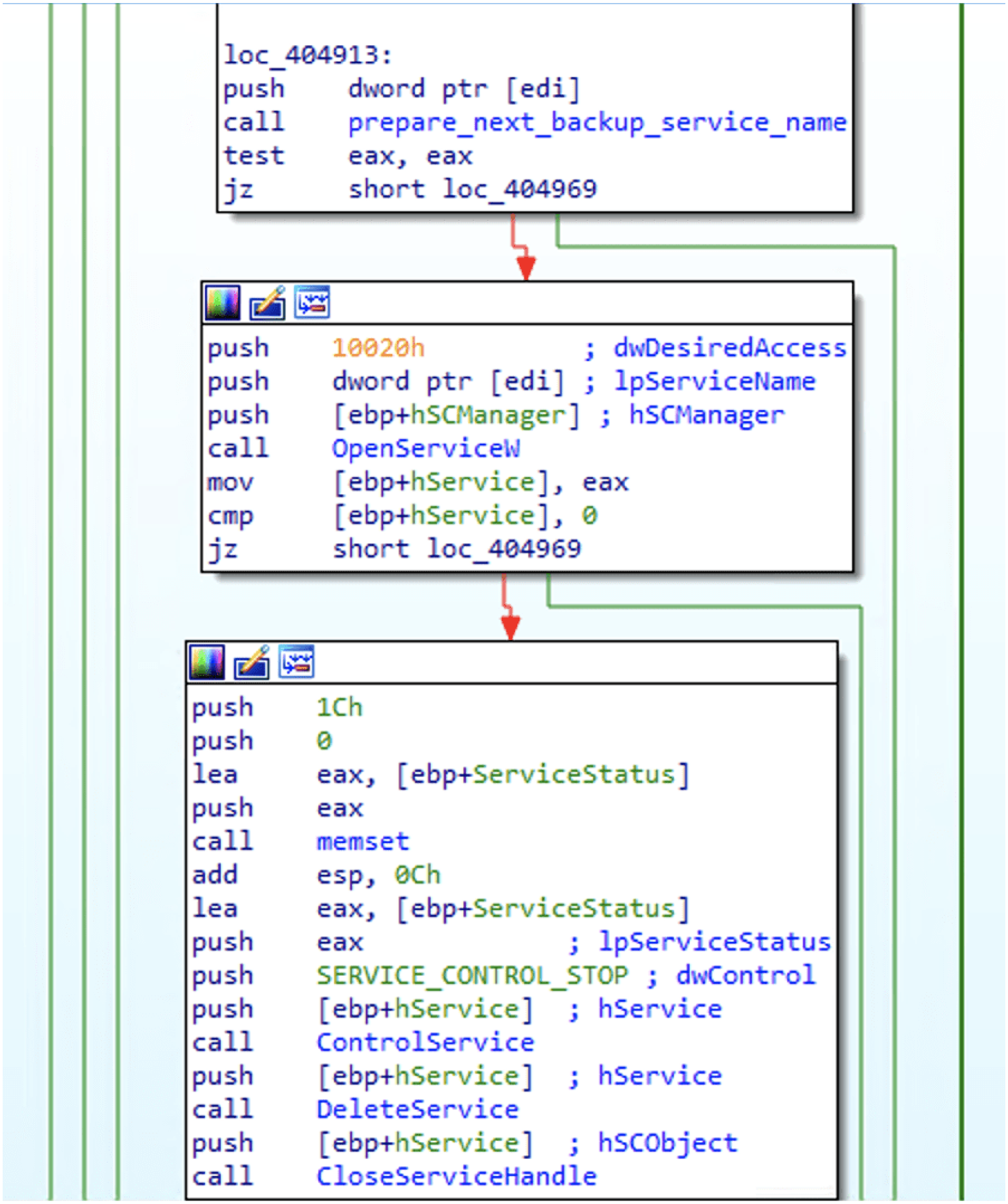


The ransomware's search for and deletion of backups.

Finally, DarkSide attempts to disable various backup solutions, searching for them by name.

```
vss sql svc$ memtas mepocs sophos veeam backup GxVss GxBlr GxFWD GxCVD GxCIMgr
```

The list of services to terminate from the embedded configuration.



The process DarkSide uses to stop and delete backup-related services.

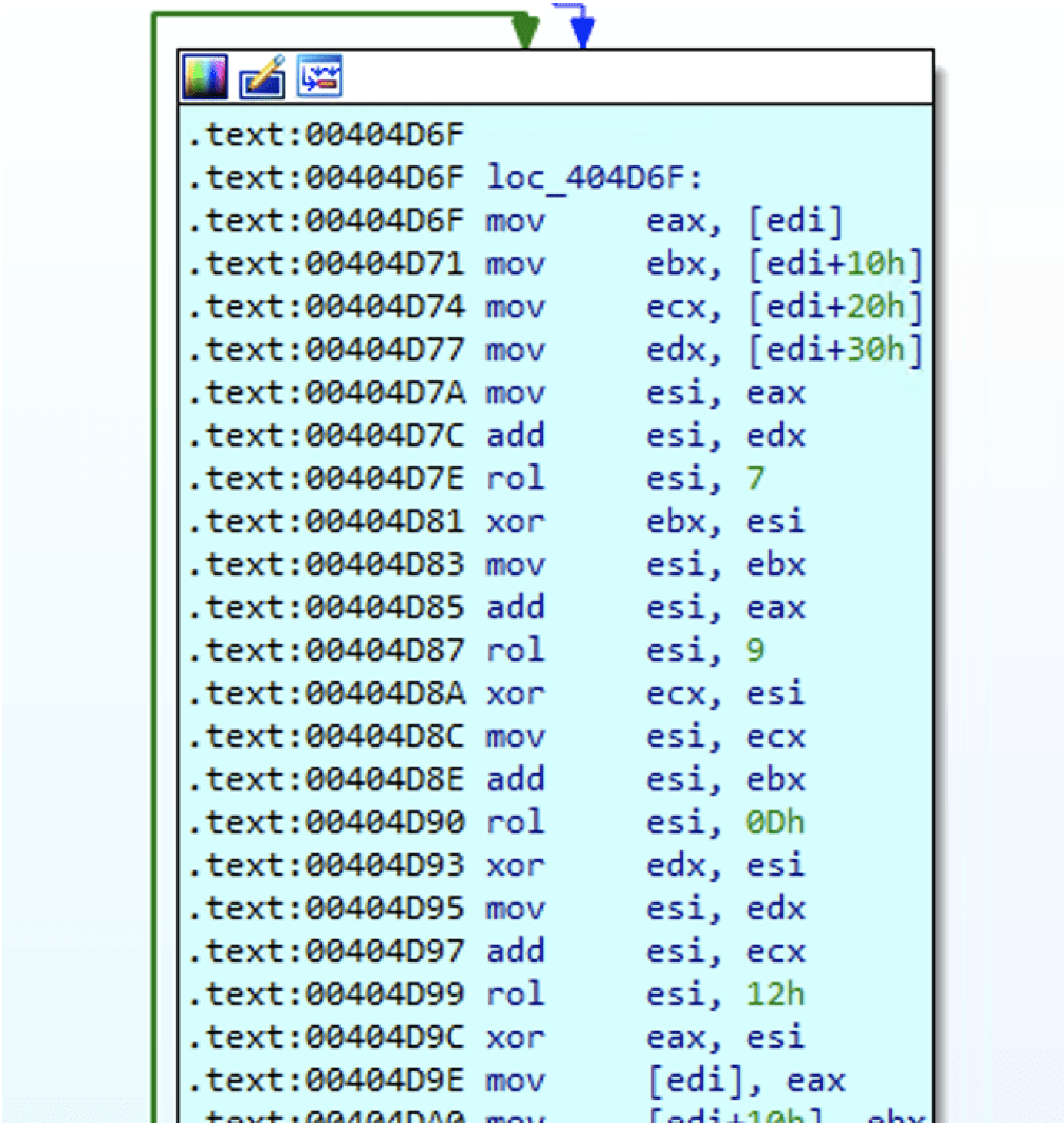
Correct Use of Symmetric and Asymmetric Encryption

Many first generations of ransomware lacked proper encryption, which made it possible for victims to recover files on their systems for free. Unfortunately, those days are long gone, and modern malware families do not repeat this mistake.

The main difference now is that symmetric encryption has been enhanced with focused use of asymmetric encryption. The former uses the same secret key for both encrypting and decrypting the data, therefore intercepting it is enough to restore access to the data.

On the other hand, asymmetric encryption uses a notion of private and public keys. While the encryption is done using a public key, the decryption is impossible without a private key. DarkSide malware implements this functionality properly by only embedding the public key in the malware and keeping the private key confidential.

The main disadvantage of asymmetric encryption over symmetric is the encryption speed. To get the best of both worlds, the authors of DarkSide encrypt victims' files using a symmetric encryption algorithm (Salsa20 with a custom matrix) and then encrypt the corresponding symmetric keys with their asymmetric public key (RSA-1024).



The symmetric Salsa20 encryption algorithm with a custom matrix.

DarkSide Demonstrates Modern Ransomware Techniques

DarkSide is just one example of a modern ransomware family that combines multiple time-tested techniques to achieve its goal. It also highlights the effectiveness of the RaaS model, which is gaining in popularity. With this model, multiple parties are involved in each attack, with a division of effort that plays to the strengths of each party.

With RaaS, experienced malware writers focus on the development of the core ransomware code, leaving deployment to affiliates who specialize in gaining access to the networks of targeted organizations. In the case of DarkSide, it is estimated that their more than 40 victims have paid $90 million in total bitcoin, with $15.5 million going to the development group and $74.7 million going to its affiliates.[3]

We hope that this technical analysis of DarkSide helps you better understand ransomware techniques and evaluate your own defenses and incident response capabilities. And, to help you detect DarkSide, IoCs and a script for decrypting embedded strings is provided at the end of this article.

It goes without saying that using network monitoring tools that help you detect unusual behavior and activity early in the malware kill chain gives you the best chance to contain ransomware before the final payload is executed. Such tools also provide actionable forensic information, as well as logs and pcaps, to assist with a timely response.

For more information on ransomware, don't miss our 20-minute webinar "Demystifying the Colonial Pipeline Attack & How Ransomware Works" and our latest "OT/IoT Security Report."

Related Content

ON-DEMAND WEBINAR

Demystifying the Colonial Pipeline Attack & How Ransomware Works

Learn about:

- How the attack happened and who was responsible
- Who DarkSide is, and what cybersecurity professionals should understand about them

- What security practices you should put in place to counter ransomware
- What recommended actions you can take to prevent future ransomware incidents

Watch Now

RESEARCH REPORT

OT/IoT Security Report

What You Need to Know to Fight Ransomware and IoT Vulnerabilities
July 2021

- Why ransomware is a formidable threat
- Analysis of DarkSide, the malware that attacked Colonial Pipeline
- Latest ICS and medical device vulnerability trends
- Why P2P security camera architecture threatens confidentiality
- How security cameras are vulnerable
- Ten measures to take immediately to defend your systems

Download

IOCs

- `0a0c225f0e5ee941a79f2b7701f1285e4975a2859eb4d025d96d9e366e81abb9`
- `baroquetees[.]com`

- rumahsia[.]com

Script

Here is an IDAPython script to decrypt embedded strings, it requires the cursor to stay at the decryption routine:

```
# Author: Nozomi Networks Labs
import idautils
import idaapi
import idc
import struct
def is_utf16_heur(string):
    counter = 0
    for val in string:
        if val == '\x00':
            counter += 1
    if counter/float(len(string)) > 0.4:
        return True
    return False
def chunks(lst, n):
    for i in range(0, len(lst), n):
        yield lst[i:i + n]
def decrypt_block(enc_string, key_matrix):
    dec_string = []
    for enc_block in chunks(list(enc_string), 255):
        temp_key_matrix = key_matrix.copy()
        bl = 0
        for i in range(len(enc_block)):
            bl = (bl + temp_key_matrix[i+1]) & 0xFF
            al = temp_key_matrix[i+1]
            ch = temp_key_matrix[bl]
            temp_key_matrix[bl] = al
            temp_key_matrix[i+1] = ch
            al = (al + ch) & 0xFF
            al = temp_key_matrix[al]
            enc_block[i] = enc_block[i] ^ al
        dec_string += enc_block
    dec_string = ''.join(map(lambda x: chr(x), dec_string))
    return dec_string
def guess_encoding(dec_string):
    utf16_flag = False
    if is_utf16_heur(dec_string):
        try:
            dec_string_print = dec_string.encode('latin-1').decode('utf-
16le')
```

```
            idc.set_inf_attr(INF_STRTYPE, STRTYPE_C_16)
            utf16_flag = True
        except Exception as e:
            pass
    if not utf16_flag:
        dec_string_print = dec_string
        idc.set_inf_attr(INF_STRTYPE, STRTYPE_C)
    # dec_string_print = dec_string_print.replace('\r',
'\\r').replace('\n', '\\n')
    return dec_string_print
def decrypt_all(enc_func, key_matrix):
    for ref in idautils.CodeRefsTo(enc_func, True):
        arg_addr = idc.prev_head(ref)
        if idc.print_insn_mnem(arg_addr) == 'push':
            enc_string_addr = idc.get_operand_value(arg_addr, 0)
            if enc_string_addr == 0:
                print('Warning: wrong address of the encrypted string at
%x: %x' % (arg_addr, enc_string_addr))
                continue
            enc_string_size = struct.unpack('<I',
idc.get_bytes(enc_string_addr-4, 4))[0]
            if enc_string_size < 0xFFFF:
                enc_string = idc.get_bytes(enc_string_addr,
enc_string_size)
            else:
                print('Warning: excessively long encrypted string at %x -
%x' % (arg_addr, enc_string_addr))
                exit(1)
            dec_string = decrypt_block(enc_string, key_matrix)
            dec_string_print = guess_encoding(dec_string)
            print('%x: %s' % (enc_string_addr, dec_string_print))
            idaapi.patch_bytes(enc_string_addr, dec_string.encode('latin-
1'))
            idc.create_strlit(enc_string_addr,
enc_string_addr+enc_string_size)
        else:
            print('Warning: non-standard argument at %x: %x' % (ref,
arg_addr))
print('Start decryption')
with open('c:\\work\\key_matrix.bin', 'rb') as fi:
    key_matrix = list(fi.read())
decrypt_all(idc.get_screen_ea(), key_matrix)
print('Done!')
```

Alexey Kleymenov

Sr. Cyber Threat Analyst

Alexey Kleymenov performs in-depth research for emerging threats and designs and develops threat intelligence infrastructure in his role at Nozomi Networks. He is passionate about reverse engineering, prototyping, process automation and research. His background includes 12+ years of practical experience with several anti-virus companies and he is a member of the (ISC)² organization.

Alexey is the author of the book "Mastering Malware Analysis: The complete malware analyst's guide to combating malicious software, APT, cybercrime, and IoT attacks."