LockBit 3.0 Ransomware Unlocked

By Dana Behling

Published: 2022-10-15 · Archived: 2026-04-05 18:17:10 UTC

Behavioral Summary

LockBit 3.0 seems to love the spotlight. Also known as LockBit Black, this ransomware family announced itself in July 2022 stating that it would now offer the data of its nonpaying victims online in a freely available easy-to-use searchable form. Then in July, it introduced a bug bounty program to find defects in its ransomware. The group even offered money to people willing to get the LockBit logo tattooed on their bodies. Regardless of the public spotlight LockBit continues its rise to the top of the ransomware ecosystem and, according to The Record, is currently the most prevalent ransomware strain.

LockBit 3.0 is a challenge for security researchers because each instance of the malware requires a unique password to run without which analysis is extremely difficult or impossible. Additionally, the malware is heavily protected against analysis and makes use of a substantial number of undocumented kernel level Windows functions.

However, in September 2022 Twitter user @3xp0rtblog announced that the builder for the ransomware was leaked by @ali_qushji and available for download from GitHub.

Figure 1: LockBit 3.0 Builder Leaked on Twitter

This leaked source allows for complete and unhindered analysis, but unfortunately also means that many new groups are emerging, using the same or modified versions of LockBit 3.0 originating from this builder.

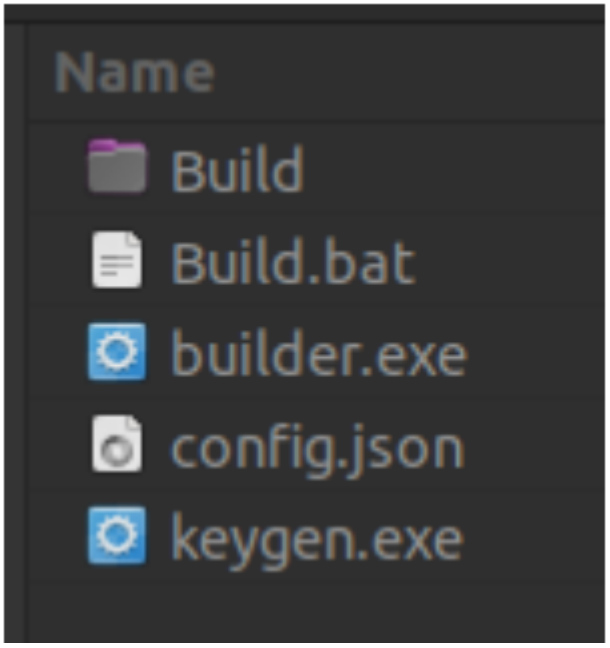


Figure 2: LockBit 3.0 Builder

The builder, once extracted, contains the files shown in Figure 2, with the Build directory empty. Running the Build.bat file, which includes the contents shown in Figure 3, automates the build process and populates the Build directory with a unique instance of the ransomware. The resulting files are shown in Figure 4.

Build.bat

```
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
```

Figure 3: Content of Build.bat

The commands that make up Build.bat, clear the Build directory, and then call keygen to generate the public and private encryption keys. The next lines that start with “builder”, generate the different variations of the LockBit 3.0 ransomware by supplying the builder with different command line options.

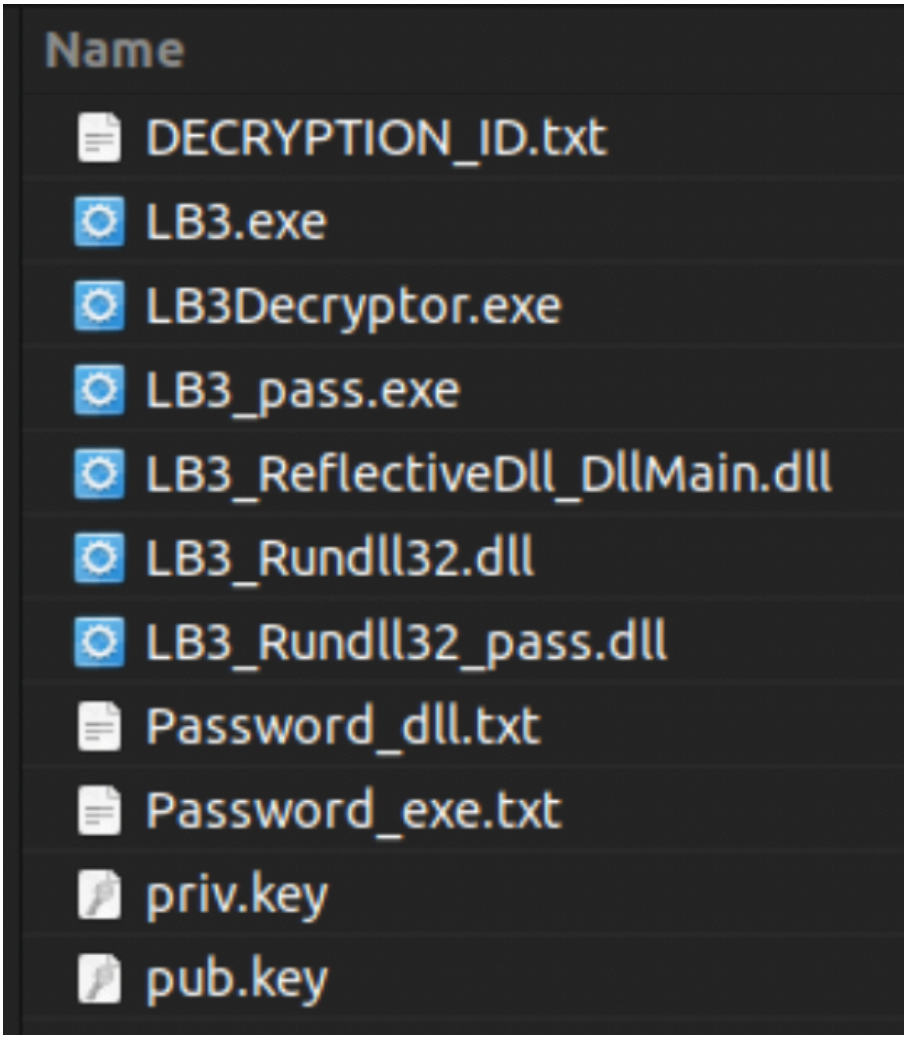


Figure 4: LockBit 3.0 File Listing after Build

Description of Generated Files

- DECRYPTION_ID.txt – Text file containing a 16-character victim ID made from the first eight hex bytes of the public key that is used to uniquely identify a victim
- LB3.exe – Compiled ransomware, which doesn't require a password
- LB3Decryptor.exe – Decryptor for the ransomware, which works with all the variations here
- LB3_pass.exe – Same as LB3.exe however requires a password to run. The password and instructions are found in Password_exe.txt in this directory
- LB3_RelectiveDLL_DLLMain.dll – Version of the ransomware that is meant to be reflectively loaded and executed in memory
- LB3_Rundll32.dll – DLL version of ransomware, which doesn't require a password.
- LB3_Rundll32_pass.dll – DLL version of ransomware, which requires the password found in the Password_dll.txt file
- Password_dll.txt – Contains password and instructions for using LB3_Rundll32_pass.dll
- Password_exe.txt – Contains password and instructions for using LB3_pass.exe
- priv.key – A private encryption key unique to this build that is used to encrypt victim files

- pub.key – A public encryption key unique to this build that is used generate various strings that tie this instance of the ransomware to a victim.

keygen.exe

The first executable called in Build.bat is keygen.exe. It generates a unique public-private key pair for each build as well as the decryption ID. Keygen appears to rely heavily on MIRACL, which according to its own description is, "…a C software library that is widely regarded by developers as the gold standard open-source SDK for elliptic curve cryptography." The generated keys are base64 encoded and saved to files priv.key and pub.key. Then the unencoded first eight bytes of the public key are converted to their hex values and saved to the DECRYPTION_ID.txt file, which acts as a unique victim identification number.

builder.exe

As shown earlier in Figure 4, the builder.exe file creates two executables, three dynamic link libraries, and two text files. In order to complete this, it requires the existing config.json (Figure 2), and the priv.key/pub.key files generated in the previous step by keygen.exe and shown in Figure 4. The builder.exe file itself also holds essentials. In its resource section are four executable template files. Each of these are used to construct the DLL and EXE encryptors as well as the program used for decryption.

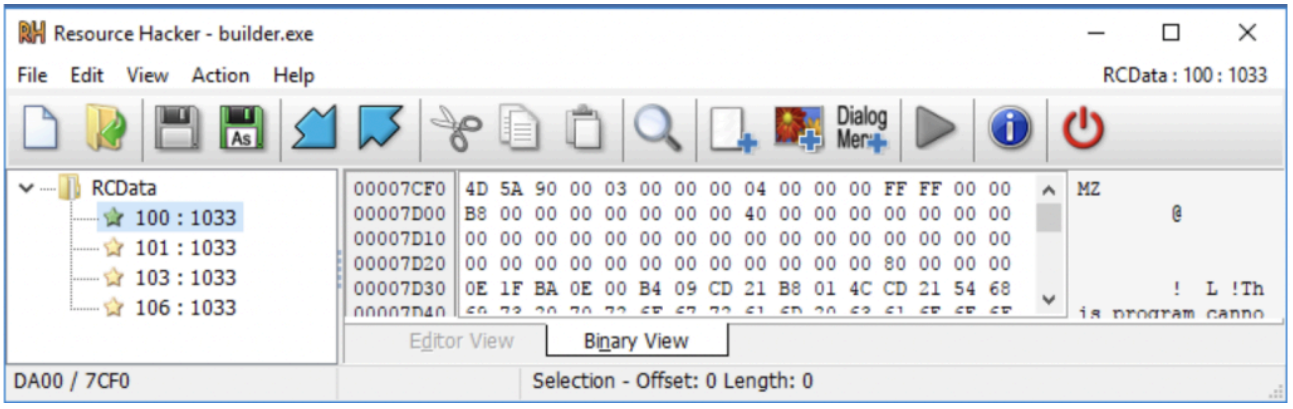
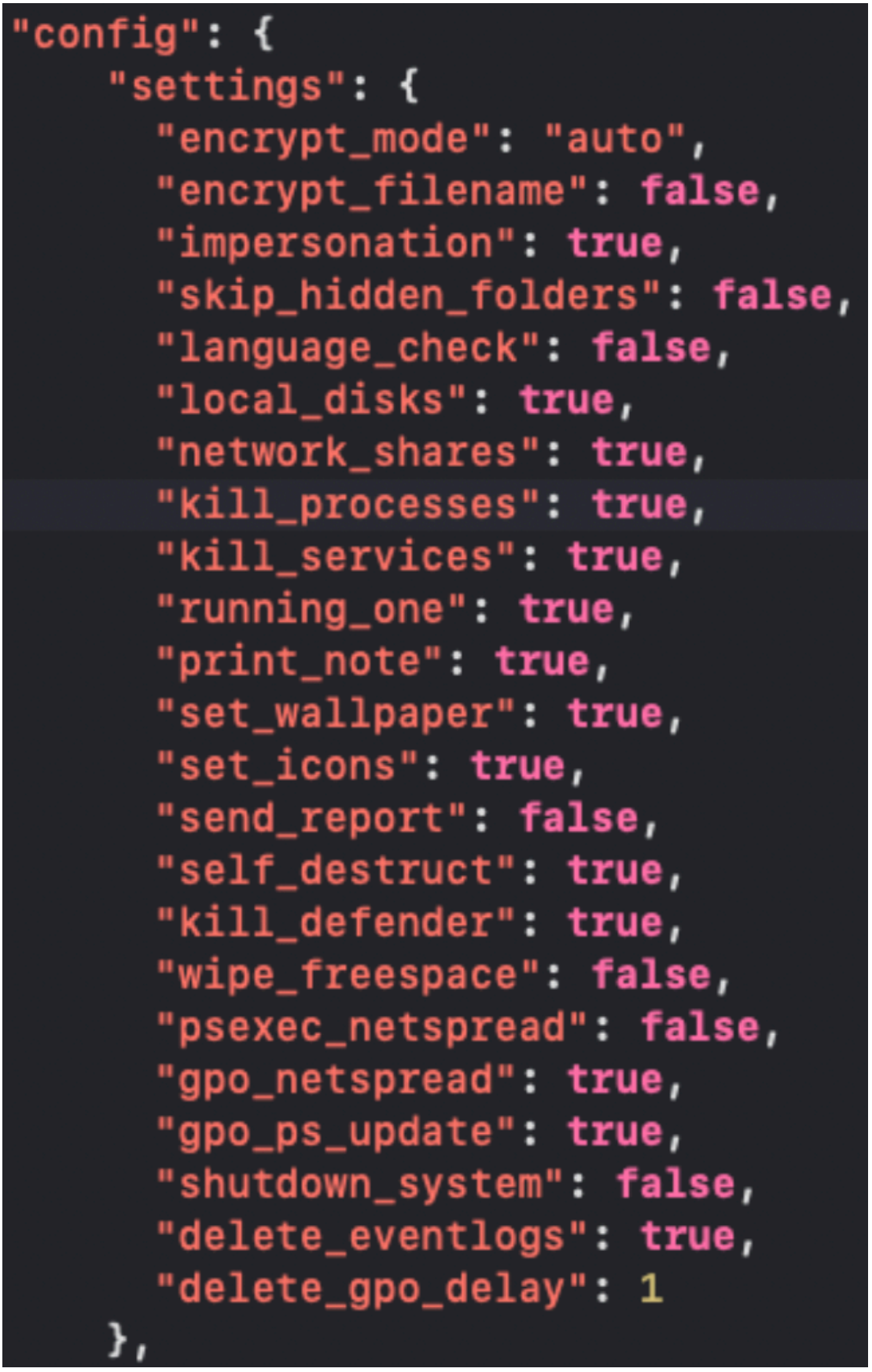


Figure 5: Builder.exe Resources

Description of each resource by its ID:

- 100 – decryptor template file
- 101 – executable template file
- 103 – DLL template file
- 106 – DLL template file that enables reflective loading

The configuration file, config.json, contains options commonly associated with ransomware including targeted folders, files to avoid, and processes that should be killed. As shown in Figure 6, it also contains settings options to tune the behavior of the ransomware, those shown below were found by default in the builder.



```
"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
    },
```

Figure 6: Configuration Options

This builder configuration allows the resultant ransomware to be tuned for a specific target environment. In addition to these configurations there are options for

- Hosts, files, folders and file extension to exclude
- Process and services to stop and remove
- List of command and control domains, URLs, or IPs
- List of usernames and passwords to try on affected systems

Since this ransomware is highly configurable there are many different code paths possible. For the sake of simplicity, the analysis here will focus on running the ransomware in the default case, without command line configuration options. However, to explain full functionality these options will be mentioned in relevant code sections.

Inside the LockBit Black Box

The initial set of code, decompiled for readability, within the program is shown in Figure 7. This code shows that the executable contains a substantial amount of unused GUI related code listed after a call to exit the process, highlighted as kernel32.ExitProcess(). All the ransom functionality occurs in the three functions that occur immediately before the call to ExitProcess. The main function contains the largest part of the ransomware functionality. The implementation of unused code is not new to malware development, but it is also not very common. A technique like this helps obscure the true purpose of the malware from researchers, and can help make malware seem legitimate.

Figure 7: Entry Function

Typical applications call DLL functions directly, or perform simple calls to get the process address of functions from the DLL file. The LockBit builder has a different and more obscure method to discover and use external functions. The function we've named "prepare_address_table_lookups", shown in Figure 7, contains a table of hashes pre-generated from a set of function calls. Each entry in this set includes the lowercase library and function names separated by a dot (i.e. ntdll.findfirstfileexw). LockBit 3.0 manually finds the DLLs it needs in the Windows System32 folder, and then manually loads each function, laboriously hashing and matching each functions library and name combination to create the address table. Once the table of hashed libraries and functions is created, the malware is careful to release and clear all memory to inhibit analysis. It should be noted that this is only a subset of the functions called, as more functions are retrieved and called from the Windows kernel during runtime.

As shown in Figure 7, the "check_priv_elevate_if_needed" function does what the name implies. Privilege escalation is achieved by duplicating access tokens to gain membership in privileged groups. To understand this it

is helpful to have a high-level understanding of Windows privileges. In life Windows privilege is roughly equivalent to a large ring of keys that provide access to otherwise restricted areas. In Windows these keys are called access tokens and each access token can contain one or more privilege constants. Each privilege constant defines a specific action or set of actions that can be restricted in the operating system. For example, one privilege constant almost everyone has is called SeShutdownPrivilege, which allows a local user to shutdown the operating system. Without this privilege constant in the access token, Windows would not allow the shutdown. Just like in real life where a physical key ring can hold numerous keys, numerous privilege constants can belong to single access token.

Essential for understanding is that processes and services started by the user or on the user's behalf are endowed with or inherit their privileges from the user. However, not all processes and services on a system belong to the user, some belong to the operating system. These operating system processes and services will by necessity have different privileges. Simply put, LockBit 3.0 duplicates an access token from a system process to use in a user process to allow functionality that would otherwise be prohibited.

To duplicate tokens LockBit 3.0 does several things. Initially it will check if it already has sufficient privilege by specifically checking for membership in the Domain Admin group. If this is not found, it attempts to grant itself a predetermined list of fifteen privilege constants, most of which were not successful in testing when run as an unprivileged user. After this, if it still does not have the required privilege, LockBit 3.0 locates the operating system process explorer.exe and calls ZwOpenProcessToken directly to read the access token for explorer.exe. Then it calls NtDuplicateToken passing in the newly acquired token handle to request read and write access to the extended attributes of the copy.

```
status = (*ntdll.NtOpenProcess)
                   (&ProcessHandle,MAXIMUM_ALLOWED,&security_descriptor.Sacl,&ClientId);
if (status == 0) {
                /* get token handle for process */
  status = (*ntdll.ZwOpenProcessToken)(ProcessHandle,MAXIMUM_ALLOWED,&TokenHandle);
  if (status == 0) {
    security_descriptor._0_4_ = 0xc;
    security_descriptor.Owner = param_2;
    security_descriptor.Group._0_1_ = 0;
    security_descriptor.Group._1_1_ = 0;
    security_descriptor.Sacl = FILE_READ_EA | FILE_WRITE_EA;
    security_descriptor.Dacl = 0x0;
    (*ntdll.NtDuplicateToken)
              (TokenHandle,MAXIMUM_ALLOWED,&security_descriptor.Sacl,0,token_type,
               &NewTokenHandle);
```

Figure 8: Duplicate the explorer.exe Process Token

In this case the Explorer process was targeted directly. However, as the program runs, each time certain processes are encountered the token privileges are checked and, if additional privileges are found, the token is duplicated. There is a noted emphasis on gaining a token with privileged domain access.

One of the first things that the malware in the main function does is check for, and create, a synchronization mutex. If the mutex is present the process will exit, ensuring that only one instance of the ransomware runs at a time. The mutex is made by first taking the MD5 hash of the supplied public key found in the pub.key file. The resulting MD5 hash is string formatted with "{%08X-%04X-%04X-%02X%02X-%02X%02X %02X %02X

%02X %02X}”. Next, the MD4 hash of this formatted string is calculated, and the result is passed to the format string “Global\%.8x%.8x%.8x%.8x”. For example, a finished mutex value would be similar to Global\ea4ee28880136cbc44dff4ad5a53561f.

Next LockBit 3.0 checks that the operating system started normally. If Windows started in safe mode the ransomware does not run most of its functionality, but instead sets a registry key to run on the next normal boot.

With the mutex in place and the boot type satisfied, the main purpose of the executable is started in a multi-threaded manner. Each thread takes on the characteristic of its tasking. Some run continuously for the life of the process, and others run once only temporarily to perform a specific task.

Stop Services: Windows Defender

The first thread removes Windows Security Services. This is achieved by taking advantage of the Trusted Installer service. The Trusted Installer, usually known by its display name “Windows Modules Installer”, is normally used for downloading and installing Windows updates and optional components. If the Trusted Installer is not already running it is started, and then its access token handle is duplicated to allow the current thread to have access to most other running services. With this token LockBit 3.0 enumerates the running services and any service matching one of those predefined in config.json is stopped and deleted. Once complete, the thread exits and does not start again.

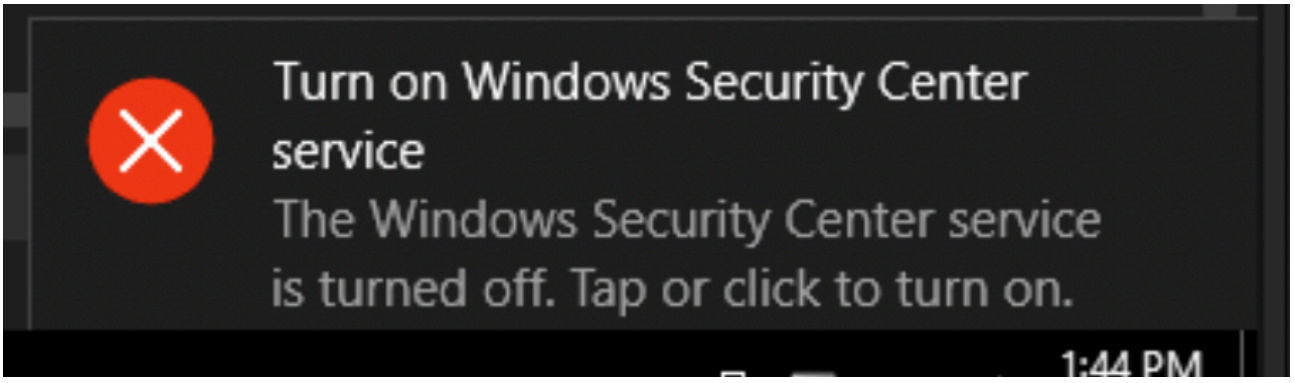


Figure 9: Notification Windows Security Center stopped

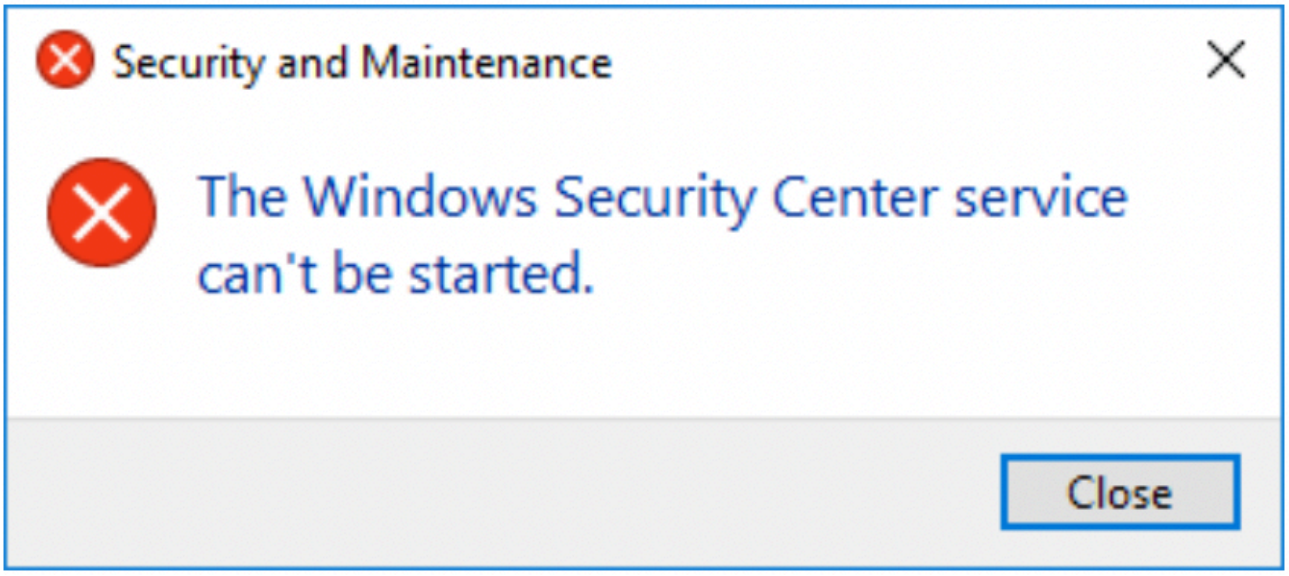


Figure 10: Error attempting to restart Windows Security Center

During testing a small taskbar notification briefly appeared to inform the user that the Windows Security Center service stopped. However, because the service was not only stopped but also deleted, the Windows notification could not be used to restart the service.

Service	Description
SecurityHealthService	Provides current information about the protection status of the endpoint, includes monitoring of Windows and other vendor's tools
wscsvc	Windows Security Center Service
Sense	Windows Defender Advanced Threat Protection Service
sppsvc	Microsoft's Software Protection Service – Licensing
WdBoot	Windows Defender ELAM (Early Launch Antimalware) Driver
WdFilter	Windows Defender Mini-Filter Driver
WdNisDrv	Windows Defender Antivirus Network Inspection System Driver
WdNisSvc	Windows Defender Network Inspection Service
WinDefend	Windows Defender Service

Table 1: Services Stopped and Deleted

It is important to consider that the Windows Security Service was targeted here because these services were listed in the default config.json file. It is highly likely that other services would be stopped if listed by threat actors using this builder.

```
          0x10020 - FILE_TRAVERSE (0x20) and DELETE (0x10000) */
hService = (*advapi32.OpenServiceW)(hSCManager,*service_name,0x10020);
if (hService != 0) {
  (*ntdll.memset)(out_lpServiceStatus,0,0x1c);
  (*advapi.ControlService)(hService,SERVICE_CONTROL_STOP,out_lpServiceStatus);
  (*advapi32.DeleteService)(hService);
  (*advapi32.CloseServiceHandle)(hService);
}
```

Figure 11: Services stopped and deleted

Once the services are removed the malware will launch several additional threads. The screenshot in Figure 12 shows the threads with the longest runtime. The first priority is to launch the thread that handles files in the Windows Recycle Bin, followed by threads to monitor for and terminate the SQL process. Then there is a thread dedicated to writing ransom notes to directories. The last threads initiated are those to encrypt files. There are three threads devoted to this in the screenshot below, but the number of threads is dynamic and will increase or decrease depending on the available system resources, and the number and type of items queued for encryption. For example, network resources identified for encryption are handled separately from those on the local system.

Number	Name	Entry	EIP
Main	Main Thread	00DF946F	7748231C
1		7744A3F0	774839BC
2		7744A3F0	774839BC
3		7744A3F0	774839BC
5		7744A3F0	774839BC
6	write over recycle bin files	00DE7468	00D50016
6		7744A3F0	774839BC
8	monitor and terminate SQL process	00DE7E58	774820AC
9	Encrypt	00DEDE78	77481DDC
10	Encrypt	00DEDE78	77481DDC
12	Encrypt	00DEDE78	77481DDC
13		7744A3F0	774839BC
13		7744A3F0	774839BC
14		745759E0	7748231C
15	write ransom notes	00DEF308	006E1852

Figure 12: Some of the LockBit 3.0 Threads

Recycle Bin Thread

As shown above, there is a special thread dedicated to handling files found in the recycle bin. The files in the recycle bin are not encrypted; instead, the content of each file is replaced with randomly generated bytes in 0x10000 byte blocks, and then they are deleted. For this reason, all files in the recycle bin are not recoverable, even with the decryptor.

Monitor and Terminate SQL Process

A separate thread runs continuously, watching for and stopping any SQL process. It does this by getting a list of services every two seconds. Each service name is passed to the function isSQL(), which looks for any occurrence

of the case insensitive string SQL. If the name contains this sequence, it is terminated. Unlike many of the other threads, this thread runs for the life of the process ensuring that SQL will not run for more than two seconds while LockBit 3.0 performs ransom activity.

```
do {
  dVar1 = *services;
  if ((services[0xf] != 0) && (success = isSQL(services[0xf]), success != 0)) {
    ClientId.UniqueProcess = services[0x11];
    ClientId.UniqueThread = 0x0;
    ObjectAttributes.Length = 0x18;
    ObjectAttributes.RootDirectory = 0x0;
    ObjectAttributes.ObjectName = 0x0;
    ObjectAttributes.Attributes = 0;
    ObjectAttributes.SecurityDescriptor = 0x0;
    ObjectAttributes.SecurityQualityOfService = 0x0;
    success = (*ntdll.NtOpenProcess)(&ProcessHandle,1,&ObjectAttributes,&ClientId);
    if (success == 0) {
      (*ntdll.ZwTerminateProcess)(ProcessHandle,0);
      (*ntdll.NtClose)(ProcessHandle);
    }
  }
  services = services + dVar1;
} while (dVar1 != 0);
calls_heap_free(out_SysInfo);
(*kernel32.sleep)(2000);
```

Figure 13: Terminate SQL Service

Delete Shadow Copies

There is a thread dedicated to the removal of Volume Shadow Copies. A shadow copy is a snapshot of a volume that duplicates all the data that is held on the volume at one well-defined instant in time. This is completed by making a call to the IWbemProvider COM object and starting an in-process server to allow queries to the Windows Management Interface. Using this it executes the WMI query “SELECT * FROM Win32_ShadowCopy”, and then deletes each of the returned Shadow Copies.

Write Ransom Notes

The ransom note thread retrieves and decrypts the embedded ransom note that was defined in config.json. The note is written to every directory not marked for exclusion. Its file name is made up of nine alphanumeric characters followed by “.README.txt” (ie xEC9do6g6.README.txt). The unique value prefixing the “.README.txt” is the base64 encoding of the first 6 bytes of the MD5 hash of the previously generated mutex GUID. Due to this method, all ransom notes left on the system are all given the same name but are unique to that system.

Encryption

Files are encrypted using the Salsa-20 algorithm. During the encryption threads, memory containing the private key is protected with heavy use of RtlEncryptMemory and RtlDecyptMemory, which makes the private key available unencrypted in memory only for the time it is necessary.

Domain Controller Discovery

Another thread attempts to logon to the infected system with the usernames and passwords found in the configuration. The usernames and passwords found in the config.json file by default are those associated with administration accounts. If any login is successful, the token membership is evaluated for membership in a domain admin group and if so, copied. Then another thread looks for and enumerates available domain controllers, getting each of their names, and attempting a remote login with the successful username and password.

Connected Drives and Shared Network Resources

Other threads check for connected drives and network resources while giving special attention to those where operating systems are installed. In all cases these newly discovered paths are passed to a function that spawns additional encryption threads.

Network Traffic

Command and control traffic occurs over TLS 1.2 to the addresses listed in the config.json file. Unfortunately, the variables and their values are AES encrypted under this TLS layer, and the order is shuffled for each request. However, the overall format of the POST request is consistent as shown in Figure 14. Consistently observed was a POST request followed by “/?” and a long string of URL style variable=value pairs separated with an ampersand, followed by the HTTP/1.1 fields. The User-Agent string is randomized, so should not be included in signatures, but the Connection, Accept-Encoding, Content-Type and Cache-Control, fields are constant. The basic format of the data section was also consistent, but the length varied slightly.

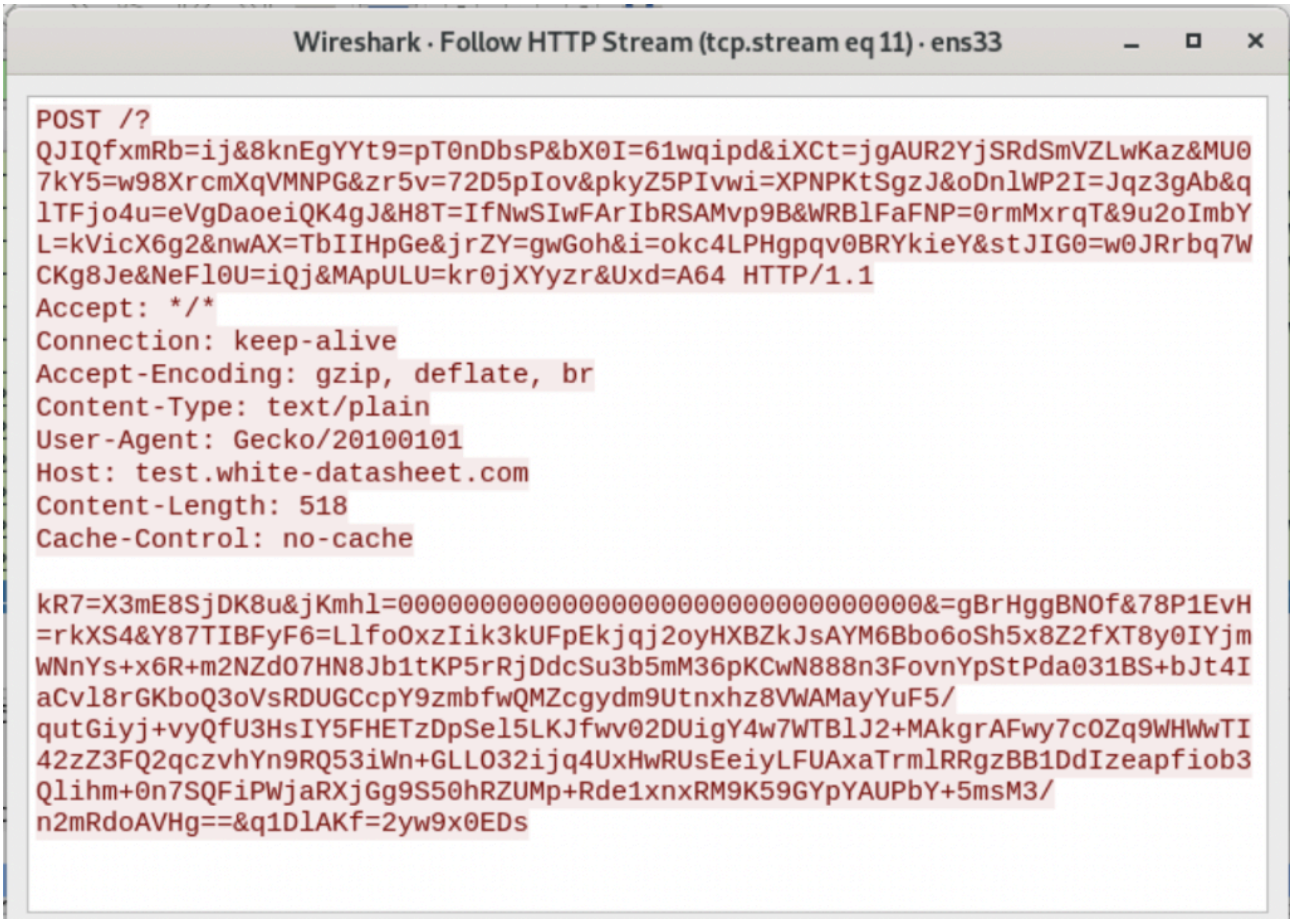


Figure 14: Initial POST Request

Although decrypting the intercepted network traffic would be close to impossible without the encryption key, the below example provides a general understanding of the information transmitted.

Figure 15: JSON Formatted Exfil Data

Post Encryption

Post encryption, the desktop background is changed to black with white text similar to the example in Figure 14. Additionally, the icons of encrypted files are changed to the LockBit “B” icon shown in the upper left corner of Figure 16.

Figure 16: Ransom Desktop and LockBit Icon

Decryptor

As the decryptor was generated with the builder, we tested its operation and use. When run, a window displays as shown below, and the user must click the large button on the right that is labeled, “Decrypt All Encrypted Files”. Once clicked the counter for All Decrypted Files ticks up as files are decrypted. As shown here, the application never closed, but by CPU usage had ceased processing files, signaling that it had completed.

There was a difference of 64 files at the end of processing. To account for these 64 files, the file system was scanned for any LockBit named files, but no matching files were found. The Recycle Bin files could account for some of the 64, but the 10 files removed from the recycle bin still left us many short of 64. These missing files aside, the Recycle Bin was not restored, and all the volume shadow copies were still erased. For all general files the decryptor did work, files were appropriately restored, and the Lockbit desktop background was removed. No conclusive tests were run to verify the number of encrypted or decrypted files with the number reported by this tool.

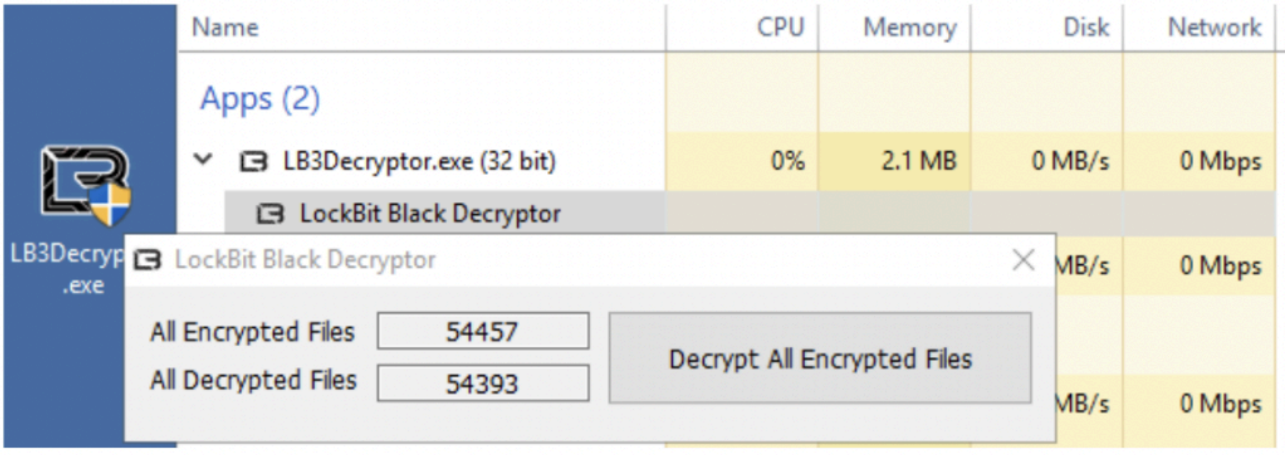


Figure 17: LockBit 3.0 Decryptor

MITRE ATT&CK TIDs

Tactic	ID	Name	Description
Execution TA0002	T1559.001	Component Object Model	Used for deleting volume shadow copies
	T1106	Native API	Copious use of Windows Native API calls
	T1047	WMI	Used for deleting volume shadow copies
Persistence TA0003	T1547.001	Registry Run Keys	If started in safe mode, sets registry to start on next normal boot.
Privilege Escalation	T1134.001	Token Impersonation	Starts processes with known token with the purpose of duplicating tokens.

TA0004			
Defense Evasion TA0005	T1562.001	Disable or Modify Tool	Stops and deletes Windows Security Services
	T1562.002	Disable Windows Event Logging	Stops and deletes service responsible for event logging
	T1562.004	Disable system firewall	Stops and deletes service for Windows Firewall.
	T1562.009	Safe Mode Boot	Changes behavior if booted in safe mode.
	T1078.001	Default Accounts	Attempts to login with default admin credentials
Discovery TA0007	T1083	File and Directory Discovery	Traverses mounted disks and file system
	T1135	Network Share Discovery	Traverses all shared network resources
	T1120	Peripheral Device Discovery	Locates removable storage devices
	T1057	Process Discovery	Looks for specific processes to stop
	T1018	Remote System Discovery	Locates domain controller and DNS server
	T1082	System Information Discovery	Gets specific information about the operating system
Lateral Movement TA0008	T1021.002	Windows Admin Shares	User of valid accounts to interact with remote network shares
Command and Control TA0011	T1071.001	Web Protocols	Uses HTTP to communicate with C2
	T1573	Encrypted channel	TLS 1.2
Exfiltration TA0010	T1041	Exfiltration Over C2 Channel	Sends basic system information in POST request

Impact TA0040	T1485	Data Destruction	Recycle bin and shadow copies are deleted
	T1486	Data Encrypted for Impact	Ransomware
	T1491.001	Internal Defacement	Desktop changed

Indicators of Compromise (IOCs)

IOC	Description
c2bc344f6dde0573ea9acdfb6698bf4c	MD5 Builder File
d6ae7dc2462c8c35c4a074b0a62f07cfef873c77	SHA1 Builder File
a736269f5f3a9f2e11dd776e352e1801bc28bb699e47876784b8ef761e0062db	SHA256 Builder File
71c3b2f765b04d0b7ea0328f6ce0c4e2	MD5 keygen File
bf8ecb6519f16a4838ceb0a49097bcc3ef30f3c4	SHA1 keygen file
ea6d4dedd8c85e4a6bb60408a0dc1d56def1f4ad4f069c730dc5431b1c23da37	SHA256 keygen file
4d388f95a81f810195f6a8dfe86be755	MD5 Resource 100
cb6fdb25a15b7797890fadc2b823984f93da5368	SHA1 Resource 100
cc3d006c2b963b6b34a90886f758b7b1c3575f263977a72f7c0d1922b7feab92	SHA256 Resource 100
87308ec0a44e79100db9dbec588260ec	MD5 Resource 101
939ff7e5eeaccb0c2f4ee080a8e403e532b6317a	SHA1 Resource 101
03b8472df4beb797f7674c5bc30c5ab74e8e889729d644eb3e6841b0f488ea95	SHA256 Resource 101
4655a7ac60ed48df9b57648db2f567ef	MD5 Resource 103
02ea524429ba2aefac63fed27e924ab3659f8c00	SHA1 Resource 103
a0db5cff42d0ee0de4d31cff5656ed1acaa6b0afab07d19f9f296d2f72595a56	SHA256 Resource 103
23a30838502f5fadc97e81f5000c4190	MD5 Resource 106
9c1142122370c9b28b13aa147c6e126b3be50845	SHA1 Resource 106
ae993930cb5d97caa5a95b714bb04ac817bcacbbf8f7655ec43e8d54074e0bd7	SHA256 Resource 106

Yara Rules

```
import "pe"
rule LockBit_3_dll
{
   meta:
         author = "VMware TAU" //bdana
         date = "2022-Oct-12"
         description = "Identifies LockBit 3.0 DLL encryptor by exported function names."
         rule_version = “1”
         yara_version = "4.2.3"
         exemplar_hash = “c2529655c36f1274b6aaa72911c0f4db7f46ef3a71f4b676c4500e180595cac6”
   condition:
      pe.exports("del") and
      pe.exports("gdel") and
      pe.exports("gdll") and
      pe.exports("gmod") and
      pe.exports("pmod") and
      pe.exports("sdll") and
      pe.exports("wdll")
}

rule LockBit_3_exe
{
   meta:
         author = "VMware TAU" //bdana
         date = "2022-Oct-12"
         description = "Identifies LockBit 3.0 exe encryptor section names, and artifact section name
         rule_version = “1”
         yara_version = "4.2.3"
         exemplar_hash = “5202e3fb98daa835cb807cc8ed44c356f5212649e6e1019c5481358f32b9a8a7”
  strings:
      $text = ".text" ascii wide
      $itext = ".itext" ascii wide
      $data = ".data" ascii wide
      $rdata = ".rdata" ascii wide
      $idata = ".idata" ascii wide
      $xyz = ".xyz" ascii wide
      $reloc = ".reloc" ascii wide
      $bss = ".bss" ascii wide
  condition:
      #text > 2 and
      #itext > 1 and
      #data > 1 and
      #rdata > 2 and
      #idata > 3 and
      $reloc and
      $bss and $xyz and not
```

```
      for any i in (0..pe.number_of_sections-1) : (
           pe.sections[i].name == ".xyz" or
           pe.sections[i].name == ".bss"
      )
}
```

Source: https://blogs.vmware.com/security/2022/10/lockbit-3-0-also-known-as-lockbit-black.html