



Sherlock

Name: Malevolent ModMaker

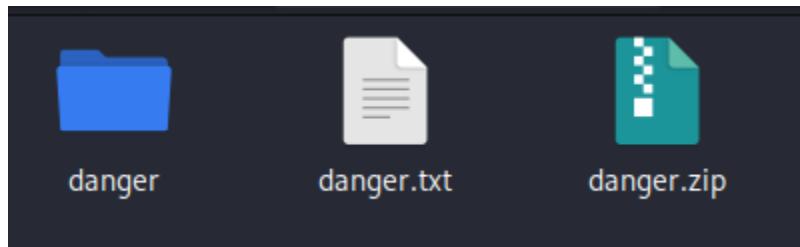
Difficulty: Medium

Category: Malware Analysis

❖ Sherlock Scenario

Bob, a senior software engineer, downloaded and executed a malicious program (MCMModMaker-v.1.4.exe) shared in his gaming community. Running it as administrator caused all his files to be deleted and replaced — classic malware behavior. The investigation revealed a ransomware attack with C2 (Command and Control) communication and encryption mechanisms.

Before we start basically, we have this file on our machine



🔍 Analysis

Task 1

Right from the start, based on the incident details, the .TXT file's contents, and the extension appended to the other .TXT file, what type of malware infection is this?

The very first clue was the deletion and replacement of files with a new extension. Combined with the ransom-related note, this strongly pointed to a classic type of malware.

This image is based on the context of the 'D.txt' file founded on the *danger* folder.

```
1 HA! GOTEEM! To get your files back, gotta pay us 1 BTC to: d32a7dafsd432789df798
2
3 We'll give you the decryption key, then you run the following command from the folder with `bruh.exe`:
4 .\bruh.exe -key <DECRYPTION_KEY>
```



Task 2

What mechanism does MCModMaker-v.1.4.exe use to send information back to the C2 server?

During static analysis of the binary, we observed outbound communication. Instead of using traditional HTTP POST requests or DNS tunneling, the program used a modern mechanism to transmit data back to its C2.

```
[old](kali㉿kali)-[~/Desktop/MalevolentModMaker(1)/danger]
└─$ strings MCModMaker-v.1.4.exe | grep "http"
  •••
```

Used strings MCModMaker-v.1.4.exe | grep "http" to extract readable text from the binary and filter for URLs, which revealed the C2 communication mechanism.

You'll see results like in this image below:

```
net/http.nop
net/http.(*persistConn).roundTrip
net/http.(transportRequest).extraHeaders
net/http.(*Request).ProtoAtLeast
net/http.(*persistConn).roundTrip.deferwrap2
net/http.(*persistConn).roundTrip.func1
net/http.(*persistConn).roundTrip.deferwrap1
net/http.(*persistConn).markReused
net/http.(*persistConn).close
net/http.(*persistConn).close.deferwrap1
net/http.(*persistConn).closeLocked
net/http.idnaASCIIFromURL
net/http.canonicalAddr
net/http.(*bodyEOFSignal).Read
net/http.(*bodyEOFSignal).condfn
net/http.(*bodyEOFSignal).Read.deferwrap1
net/http.(*bodyEOFSignal).Close
net/http.(*bodyEOFSignal).Close.deferwrap1
net/http.(gzipReader).Read
net/http.(gzipReader).Close
net/http.tlsHandshakeTimeoutError.Timeout
net/http.tlsHandshakeTimeoutError.Temporary
net/http.tlsHandshakeTimeoutError.Error
net/http.fakeLocker.Lock
net/http.fakeLocker.Unlock
net/http.(*connLRU).add
net/http.envProxyFunc.func1
```

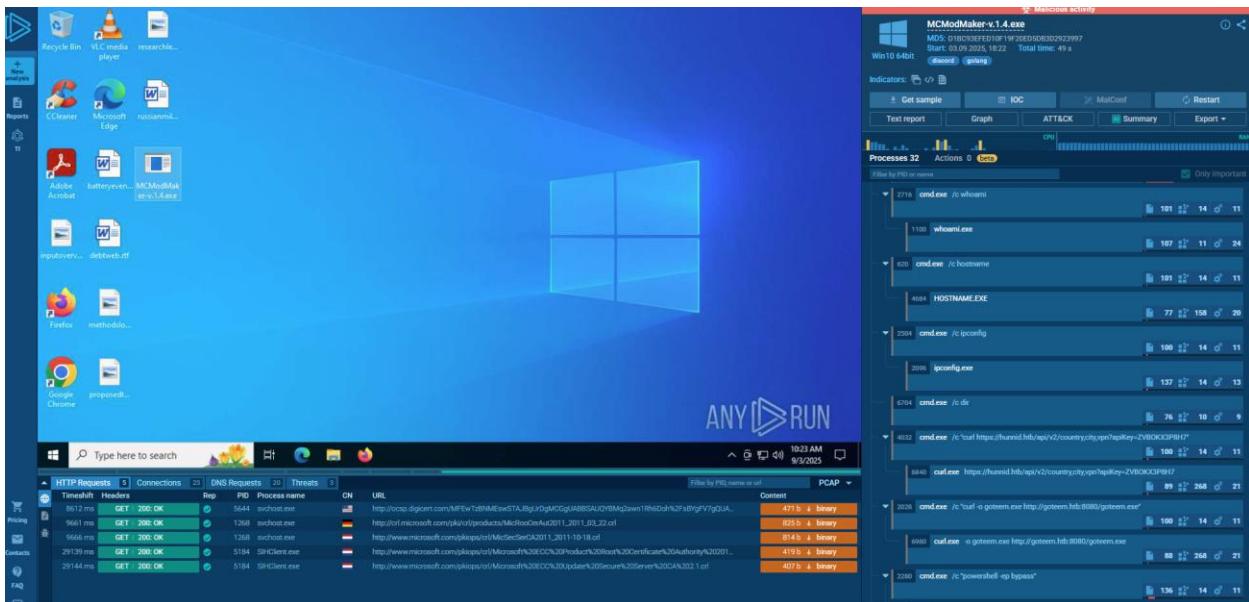


Task 3

What command does MCModMaker-v.1.4.exe run suggesting that it is meant to execute other binaries or scripts?

Generated the SHA-256 hash of MCModMaker-v.1.4.exe and uploaded it to AnyRun for dynamic analysis, where the execution command used by the malware was revealed.

```
(kali㉿kali)-[~/Desktop/MalevolentModMaker(1)/danger]
$ sha256sum MCModMaker-v.1.4.exe
c676db4315721f28da552dabf5340e92435a8726d705235a69d0639fa5c73a08  MCModMaker-v.1.4.exe
```





Task 4

What is the value of the API key contained within the URL which suggests it enumerates geolocation data?

In VirusTotal, I reviewed the Behavior tab, I spotted a URL containing an embedded key related to geolocation data.

The screenshot shows the VirusTotal interface. At the top, there is a search bar with the hash value "c676db4315721f28da552dabf5340e92435a8726d705235a69d0639fa5c73a08". To the right of the search bar are several icons: a download arrow, a refresh symbol, and a help question mark.

Below the search bar, the "Activity Summary" section is visible, featuring a "Download Artifacts" button. Under "MITRE ATT&CK Tactics and Techniques", the following items are listed:

- + Execution TA0002
- + Defense Evasion TA0005
- + Discovery TA0007
- + Command and Control TA0011

Below this, the "Malware Behavior Catalog Tree" is shown:

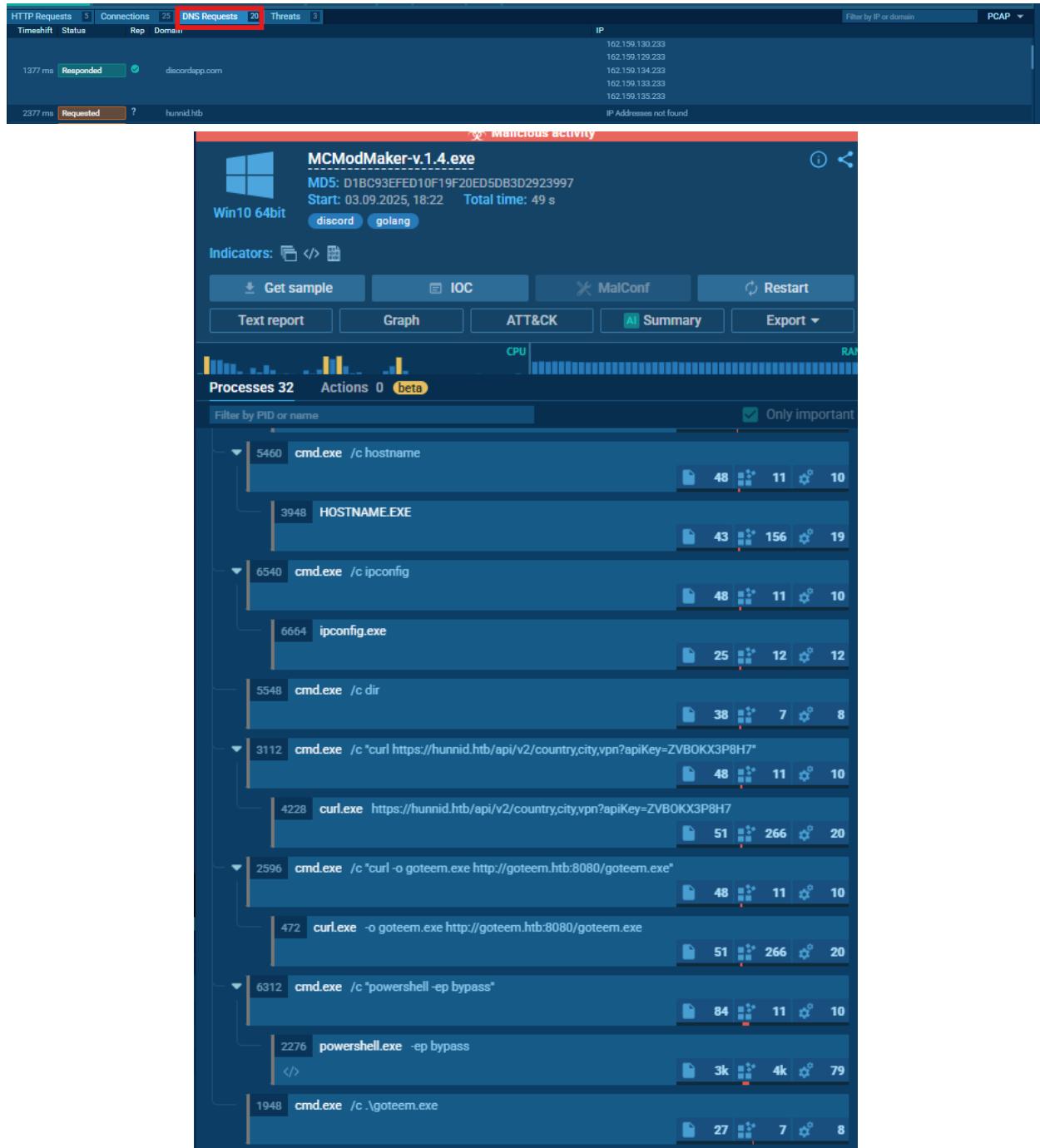
- + Anti-Behavioral Analysis OB0001
- + Anti-Static Analysis OB0002
- + Defense Evasion OB0006
- + Discovery OB0007
- + Impact OB0008
- + Execution OB0009
- + Persistence OB0012
- + File System OC0001
- + Memory OC0002



Task 5

What domain is the C2 server that serves the ransomware payload?

By reviewing DNS requests and related process activity in AnyRun, I was able to identify the domain used by the malware to host and deliver the ransomware payload.





Task 6

While analyzing the MCModMaker-v.1.4.exe, what format is the data that is returned to the C2 server?

I used the strings utility with a filter to look for indicators of structured data formats within the binary. The terminal returned to multiple matches, and scrolling through confirmed the format being used in C2 communication.

```
(kali㉿kali)-[~/Desktop/MalevolentModMaker(1)/danger]
└─$ strings MCModMaker-v.1.4.exe | grep -i "json"
```

```
ABI descriptionbad value for fieldEgypt Standard TimeSudan Standard TimeLibya Standard TimeBahia Standard TimeHaiti Sta
ard TimeYukon Standard TimeAltai Standard TimeIndia Standard TimeSyria Standard TimeRussia Time Zone 11Nepal Standard Ti
Korea Standard TimeChina Standard TimeRussia Time Zone 10Tokyo Standard TimeTomsk Standard TimeSamoa Standard TimeTonga
andard Timerevoked certificateexpired certificateunknown certificateunknown cipher typeinvalid URL escape missing ']' in
ostmultipartmaxheaderscriterion too shortskip this directorymime: no media typeevictCount overflowillegal instructionbad
ile descriptordisk quota exceededtoo many open filesdevice not a streamdirectory not emptyCryptReleaseContextGetTokenInfor
mationCreateSymbolicLinkWGetCurrentProcessIdSetTokenInformationMultiByteToWideCharunknow hash value negative coordinate
09: malformed OIDx509: trailing data509: unknown errorunsupported AEAD id2006010215040520700zero length segmentTCODENot
plementedfile already existsfile does not existfile already closedmodulus must be oddunknown Go type: %vjson: error call
g pad length too largehttpplaxcontentlengthMAX_HEADER_LIST_SIZEconnection error: %sframe_settings_mod_6conn_close_lost_pi
assigned stream ID 0read_frame_too_largeunknown address typeRequest URI Too LongUnprocessable EntityInsufficient Storage
valid write resultreflect.Value.IsZeroreflect.Value.SetInt37252902984619140625floating point errorGC sweep terminationRe
tDebugLog (test)chan send (nil chan)flushing proc cachesmalloc during signalclose of nil channelinconsistent lockedmnote
leep not on g0bad system page size to unallocated span/gc/scan/stack:bytes/gc/scan/total:bytes/gc/heap/frees:bytes/gc/go
mlimit:bytesp mcache not flushed markroot jobs done
estroyEnvironmentBlockexit hook invoked panicinvalid PrintableStringx509: malformed UTCTimex509: invalid key usagex509:
lformed versiontoo many pointers (>10)segment length too longunpacking Question.Nameunpacking Question.Typeskipping Ques
on Classpattern bits too long: flate: internal error: P224 point not on curveP256 point not on curveP384 point not on cu
eP521 point not on curveinvalid scalar encodingasn1: structure error: truncated tag or lengthjson: unsupported type: une
nected buffer len=%vinvalid pseudo-header %qframe_headers_prio_shortinvalid request :path %qread_frame_conn_error_%sReque
Entity Too Largehttp: nil Request.Header116415321826934814453125582076609134674072265625tracecheckstackownershiphash of
nhashable type span has no free objectsruntime: found obj at *(runtime: VirtualFree of /cgo/go-to-c-calls:calls/gc/heap/
jects:objects/sched/latencies:secondsqueuefinalizer during GUpdate during transitionruntime: markroot index can't scan
r own stackgcDrainN phase incorrectpageAlloc: out of memoryruntime: p.searchAddr = range partially overlapsstack trace u
available
hacha20: wrong key sizereflect.Value.OverflowIntjson: unsupported value: http: invalid cookie namehttp2: Request.URI is
ltext/plain; charset=utf-8http2: Framer %p: read %vframe_data_pad_byte_shortframe_settings_has_streamframe_headers_zero_
reamframe_headers_pad_too_bigframe_priority_bad_lengthhttp2: invalid header: %vstrict-transport-securityhttp2: unsupport
```

Task 7

What specific filetype is enumerated by goteem.exe for encryption?

First, I obtained the SHA-256 hash of goteem.exe then uploaded it to VirusTotal.

```
(kali㉿kali)-[~/Desktop/MalevolentModMaker(1)/danger]
└─$ sha256sum goteem.exe
3e867688e72f5af4f6f035c46bfd1c6fbb9adcb57b88231f0b067938d7100070  goteem.exe
```



By reviewing the file system activity in VirusTotal's behavior tab, I confirmed that the ransomware attempts to enumerate and encrypt files with a specific extension.

The screenshot shows the VirusTotal analysis interface for the file 3e867688e72f5af4f6f035c46bfd1c6fb9adcb57b88231f0b067938d7100070. The 'BEHAVIOR' tab is highlighted with a red box. The page indicates 28/71 security vendors flagged the file as malicious. The file is a 64-bit executable named goteem.exe, peexe. The community score is 28/71. The 'Behavior' section lists grouped sandbox reports from various engines like C2AE, CAPA, CAPE Sandbox, Microsoft Sysinternals, and Zenbox. Each entry includes a checkmark, the engine name, and numerical counts for various metrics.

The screenshot shows the 'Activity Summary' section of the VirusTotal analysis. It displays 'Behavior Similarity Hashes' for various engines: C2AE, CAPA, CAPE Sandbox, Microsoft Sysinternals, and Zenbox, each with its corresponding SHA-256 hash. A large red arrow points down to the 'File system actions' section, which is currently collapsed. Below it, the 'Files Opened' section is also collapsed.

Engine	SHA-256 Hash
C2AE	fe66e54118bb12b06dcabb6c2d17206
CAPA	d6e2533976cc43cabd677b9fa12ec849
CAPE Sandbox	c60377368a8063e47df4337a97d97d46
Microsoft Sysinternals	10e5b6517952bcd39a2077ee497337fa
Zenbox	8c991c8ccbce0595f041bdb8d852f92b

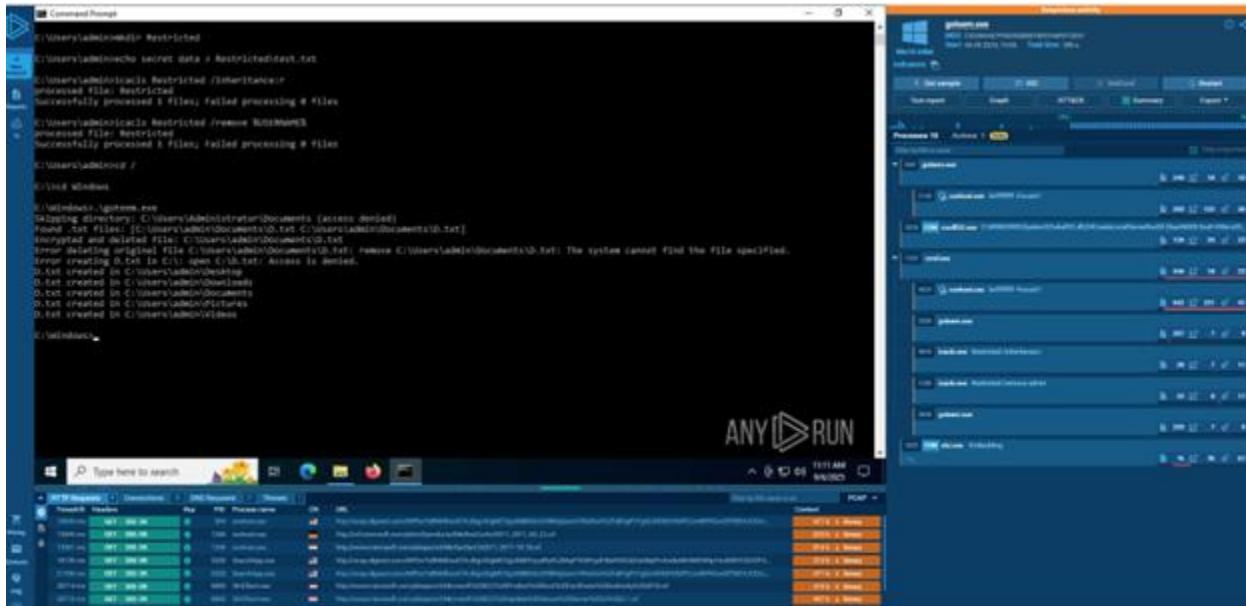


Task 8

What is the full string that's displayed when goteem.exe enumerates a restricted Windows folder?

First, I generated the hash of the binary and uploaded it to Any.Run to gather hints on its behavior. From there, I dug into the disassembly and located the function call that handles the encryption process.

```
(kali㉿kali)-[~/Desktop/MalevolentModMaker(1)/danger]
$ sha256sum goteem.exe
3e867688e72f5af4f6f035c46bfd1c6fbb9adcb57b88231f0b067938d7100070  goteem.exe
```



By running strings on the binary and filtering error messages, I found output showing how the program responds when encountering restricted Windows directories.

```
(kali㉿kali)-[~/Desktop/MalevolentModMaker(1)/danger]
$ strings goteem.exe | grep "access denied"
index out of range [%x] with length %ym changed unexpectedly in cgocallbackmakechan: invalid channel element typeunreachable method called. linker bug?concurrent map iteration and map writegcBgMarkWorker: blackening not enabledcannot read stack of running goroutineuntime: blocked read on free polldescrentime: LoadLibraryExW failed; errno=runtime: GetProcAddress failed; errno=runtime: sudog with non-false isSelectarg size to reflect.call more than 16Bv could not fit in traceByte
```



Task 9

Within the encryptFile function, in case file was read successfully, what is the function name found at the call instruction?

I used Ghidra for static analysis (though other decompilers could also work).

```
(kali㉿kali)-[~/Desktop/MalevolentModMaker(1)/danger]
$ ghidra
```

After setting up Ghidra, I filtered for encryptFile and then opened main.encryptFile. By examining the call instructions inside, I identified the specific function responsible for handling the encryption process.

The screenshot shows the Ghidra interface with the following details:

- Program Trees:** Shows the file structure of goteem.exe, including Headers, AT, rdata, data, .pdata, .xdata, .idata, and .reloc.
- Symbol Tree:** Shows the namespaces, with **main.encryptFile** highlighted.
- Listing:** The assembly listing for the **main.encryptFile** function. A search filter "encryptFile" is applied to the assembly view.
- Decompiler:** The decompiled C code for the **_rt0_and64_windows** function. The code includes imports for **cpuid**, **cpuid_Version_info**, and **cpuid_basic_info**.
- Console - Scripting:** An empty console window.

```
1
2 // WARNING: Removing unreachable block (ram.0x0046a466) /*
3 /* WARNING: Removing unreachable block (ram.0x0046a43b) */
4 /* Golang function info: Flags: [ASIM]
5 Golang source: /usr/local/go/src/runtime/r0_windows_amd64.s:10 */
6
7 void _rt0_and64_windows(void)
8 {
9     int in_GS_OFFSET;
10    undefined4 *puStack;
11    undefined4 *puVar2;
12    dword sVar3;
13    void *v;
14
15    undefined8 auStack_1030 [65520];
16    undefined8 auStack_1030 [65520];
17    undefined8 auStack_40;
18    undefined4 *puStack_38;
19    undefined4 auStack_30 [2];
20    undefined1 *puStack_28;
21    undefined4 auStack_16;
22    undefined1 *puStack_16;
23
24    puStack_10 = &auStack[0x00000000];
25    puStack_38 = &auStack_30;
26    DAT_005e4100 = auStack_1030;
27    psVar1 = (sword *)cpuid_basic_info();
28    sVar3 = psVar1[1];
29    if (*psVar1 != 0) {
30        if (((cvvar3 == 0x756e6547) && (psVar1[2] == 0x49656e69)) && (psVar1[3] == 0x005e8c5a) == 1;
31        DAT_005e4100 = auStack_1030;
32    }
33    puVar2 = (undefined4 *)cpuid_Version_info();
34    DAT_005e4100 = puVar2;
```



Using Ghidra, I switched to the decompile tab, where the relevant constant stood out during review, allowing me to confirm the key used in the process.

Cf Decompile: main.encryptFile - (goteem....)

```
1  /* Golang function info: Flags: []
2   Golang source: /root/goteem/main.go:44
3   Golang stacktrace signature: func main.encryptFile(struct? {8, 8},
4
5   void main::main.encryptFile
6       (uint8 *param_1,int param_2,undefined8 param_3,undefined8 param_4,
7        undefined8 param_5,undefined8 param_6)
8
9 {
10    string val;
11    string val_00;
12    string val_01;
13    string val_02;
14    string val_03;
15    string a0;
16    string name;
17    string val_04;
18    string val_05;
19    [l]uint8 [Var1];
20    [l]uint8 [Var2];
```

Task 10

What is the decryption key?

To uncover the decryption key, I analyzed the binary in Ghidra and reviewed the decompile tab. The constant used for decryption was clearly visible within the function logic.

Then I went back to the terminal, I used the strings utility with a keyword filter to search for references to cryptographic routines. This revealed the specific function being called within the encryption process.

The image below shows the output of my command, filtered for cryptographic function references. It aligns with what I had identified in the previous task and confirms the function call used during the encryption process.



```
1aavxfaintmapMayUTCEET+00+01CATWATEATGMTHSTHDT-03-04-05ESTCSTCDTMSTMDT-02EDTASTADTPSTPDTNSTNDT+06+07PKT+11KST+05JST+10-01-11-12-08-09+13CETBSTMKS-06+14StdDlt\\.\.\?\?nettrueopenreadfilesyncpipeallgadis LEAFbase of ) = <=>GOGC] = pc+=Inf-Inf: p=cas1cas2cas3cas4cas5cas6 at fp= gp= mp= m=3125ermssse3avx2bmi1bmi2boolint8uintchanfuncallkind on JuneJulyEESTSASTAKSTAKDTACSTNZDTbindtimeD.txtfalse<nil>ErrorwritecloseLstatdefersweptestRtestWexecWhchanexecRschedsudogtimeeep cnt=gcing MB, got= ...dval=gcstopm: negative nm spinningfindunnable: netpoll with psave on system g not allowednewproc1proc1: new g is not GdeadFixedStack is not power-of-2missing stack in shrinkstack args stack map eme symbol tableruntime: no module data for mismatched issending updates[originating from goroutine large18189894035458564758300781259094947017729282379150390625Canada Central Standard TimeCen. Aus Central W. Standard TimeCentral Europe Standard TimeEnglish name for time zone "file descriptor address requiredprotocol driver not attachedCertCreateCertificateContextabi.NewName: name too longkey size executing on Go runtime stacknotesleep - waitm out of sync/cpu/classes:idle:cpu-seconds/conds/gc/heap/allocs-by-size:bytes/gc/stack/startng-size:bytesgc done but gcphase ≠ _GCoffruntimeanobject of a noscan objectruntime: marking free object addspecial on invalid pointerruntime: summe: levelShift[level] = doRecordGoroutineProfile gp1=NtCreateWaitCompletionPacket of range1136868377216160297397988281255684341886080801486968994140625reflect: Len of non-array tStandard TimeCentral Brazilian Standard TimeMountain Standard Time (Mexico)cannot assign requested a/u/classes/user:cpu-seconds/gc/heap/allocs-by-size:bytes/gc/stack/startng-size:bytesgc done but gcphase ≠ _GCoffruntime: p.gcMarkWorkerMode= scanobject of a noscan objectruntime: marking free object addspecial on invalid pointerruntime: summary max pages = runtime: levelShift[level] = doRecordGoroutineProfile gp1=NtCreateWaitCompletionPacket unsafe.String: len out of range1136868377216160297397988281255684341886080801486968994140625reflect: Len of non-array ty pe W. Central Africa Standard TimeCentral Brazilian Standard TimeMountain Standard Time (Mexico)cannot assign requested a
```

Task 11

What is the name of the project in the encrypted .TXT file?

Now since you have the decryption key, go back to the danger folder and read 'D.txt'

```
1 HA! GOTEEM! To get your files back, gotta pay us 1 BTC to: d32a7dafsd432789df798
2
3 We'll give you the decryption key, then you run the following command from the folder with `bruh.exe`:
4 .\bruh.exe -key <DECRYPTION_KEY>SS|
```

Follow the instructions

Note: The executable has Windows format, so running it on Linux/macOS might fail unless you use Wine (*Install wine if you don't have yet*):

In my case I used *wine* since I'm on Kali Linux.

Then run bruh.exe with your decryption key

Example: *wine bruh.exe -key 123456789*

Ensure you are in the same directory as bruh.exe. Use *ls* to confirm.

After that go back to the danger folder then open *probably_important.txt* and you'll find the project name there.



```
File Edit Search View Document Help
D.txt - /Desktop/MalevolentModMaker()/.danger/probably_important.txt - Mousepad
probably_important.txt

1 INTERNAL SOFTWARE REQUIREMENTS SPECIFICATION
2 Project:
3
4 Classification: INTERNAL USE ONLY Version: 1.0 Last Updated: January 5, 2025 Author: Engineering Team - Tools & Automation Division
5 Purpose
6 This document defines the functional and non-functional requirements for the development of an internal AI-powered coding assistant chatbot. The chatbot will assist developers with code generation, debugging, documentation, and API usage across multiple programming languages and frameworks. It is intended for use within the engineering organization and is not designed for external deployment.
7 Scope
8 1. Purpose
9
10 The chatbot will be integrated into the internal developer portal and optionally exposed via CLI and IDE plugins. It will support natural language queries and return context-aware code snippets, explanations, and troubleshooting guidance. Supported languages include Python, JavaScript, C++, PowerShell, and Bash.
11 2. Intended Users
12
13 Software Engineers
14 Security Engineers
15 DevOps and SRE Teams
16 QA Automation Engineers
17 Technical Writers
18
19 3. Functional Requirements
20
21 4.1 Natural Language Interface
22
23 Accepts free-form developer queries (e.g., "How do I write a recursive function in Python?")
24 Supports follow-up questions and conversational context retention
25
26 4.2 Code Generation
27
28 Generates syntactically correct and idiomatic code snippets
29 Supports multiple languages and frameworks
30 Offers inline comments and optional documentation blocks
31
32
33
34
35
36
37
```

That's it — Malevolent ModMaker Solved.

We uncovered the ransomware behavior, analyzed its execution flow, and retrieved the decryption key. A neat exercise in malware analysis and reverse engineering.

