Sherlock

Name: Malevolent ModMaker

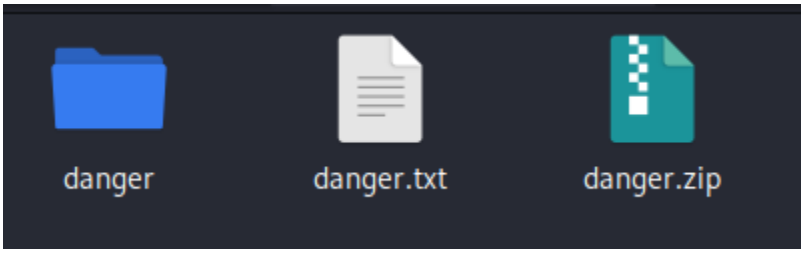
Difficulty: Medium

Category: Malware Analysis

Sherlock Scenario

Bob, a senior software engineer, downloaded and executed a malicious program (MCModMaker-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>S
```

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.

```
┌──(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:

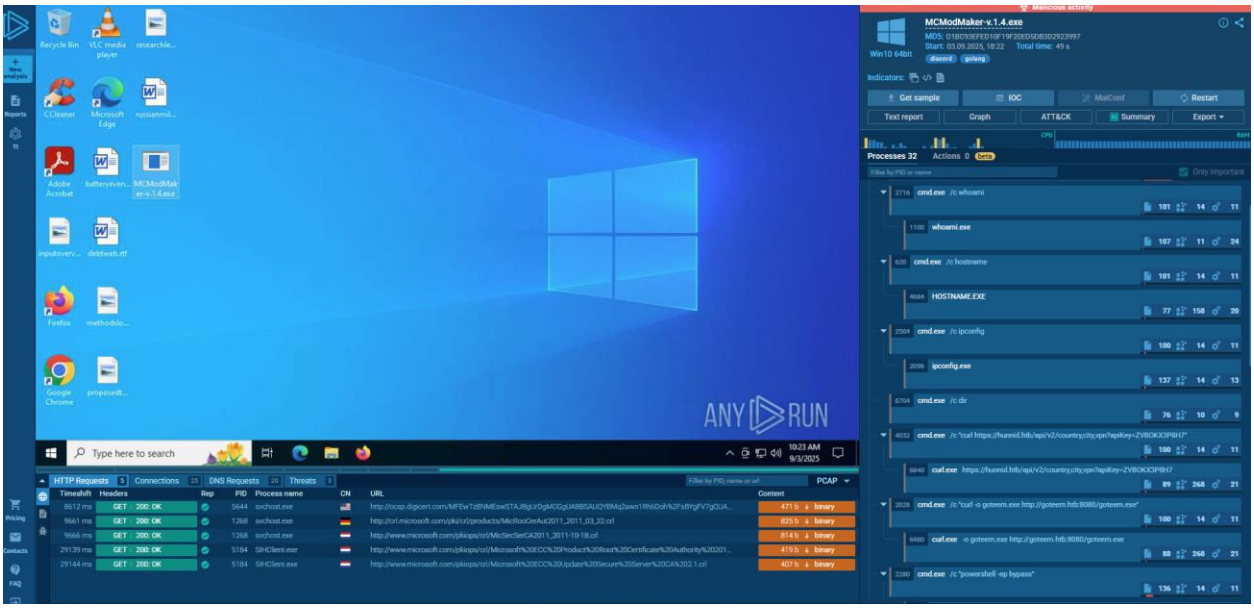
```
net/http.nop
net/http.(*persistConn).roundTrip
net/http.(*transportRequest).extraHeaders
net/http.(*Request).ProtoAtLeast
net/http.(*persistConn).roundTrip.deferwrap2
net/http.(*persistConn).roundTrip.func1
net/http.(*persistConn).roundTrip.deferwrap1
net/http.(*persistConn).markReused
net/http.(*persistConn).close
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.

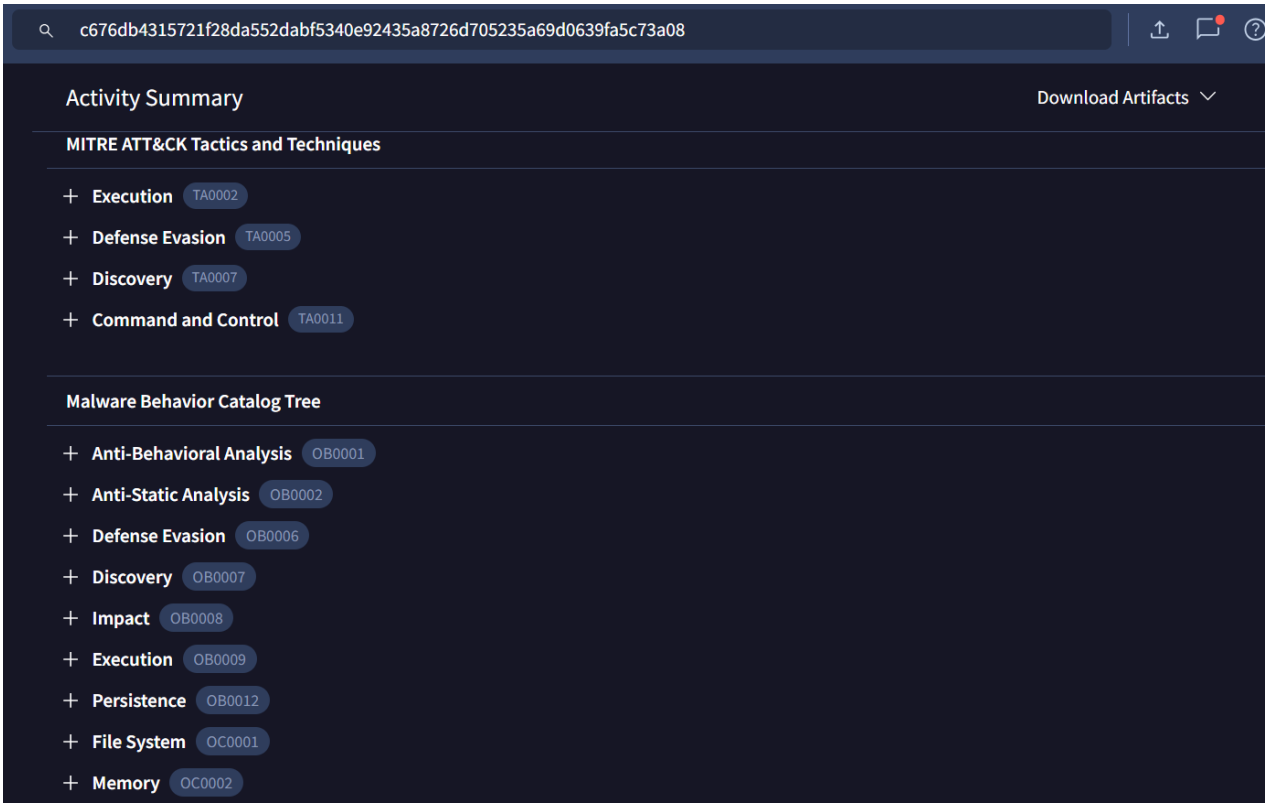
```
┌──(kali㉿kali)-[~/Desktop/MalevolentModMaker(1)/danger]
└─$ sha256sum MCModMaker-v.1.4.exe
c676db4315721f28da552dabf5340e92435a8726d705235a69d0639fa5c73a08  MCModMaker-v.1.4.exe
```



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.



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 emptyCryptReleaseContextGetTokenInf
mationCreateSymbolicLinkWGetCurrentProcessIdSetTokenInformationMultiByteToWideCharunknown hash value negative coordinate
09: malformed OIDx509: trailing datax509: unknown errorunsupported AEAD id20060102150405Z0700zero length segmentRCodeNot
plementedfile already existsfile does not existfile already closedmodulus must be oddunknown Go type: %vjson: error call
g pad length too largehttplaxcontentlengthMAX_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.SetInt372529029846191406 25floating 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
ected 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 GCupdate during transitionruntime: markroot index can't scan
r own stackgcDrainN phase incorrectpageAlloc: out of memoryruntime: p.searchAddr = range partially overlapsstack trace u
vailable
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
```

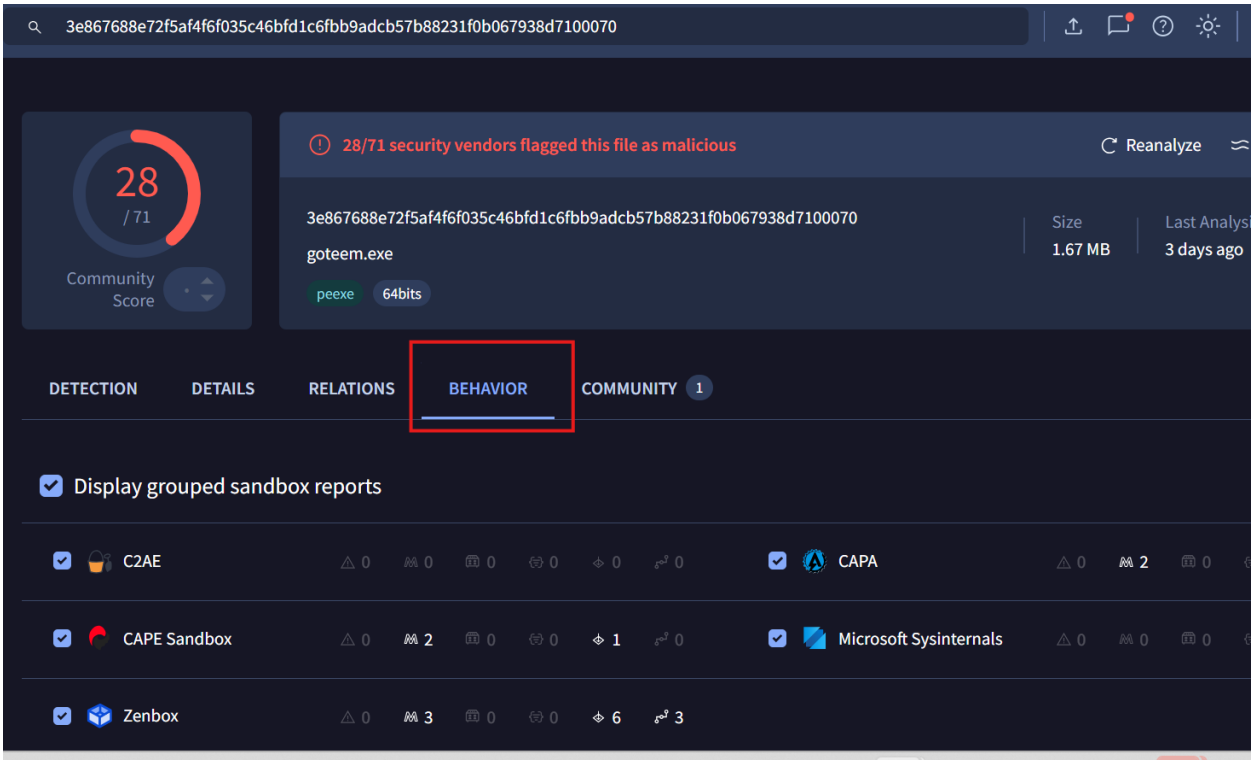
Task 7

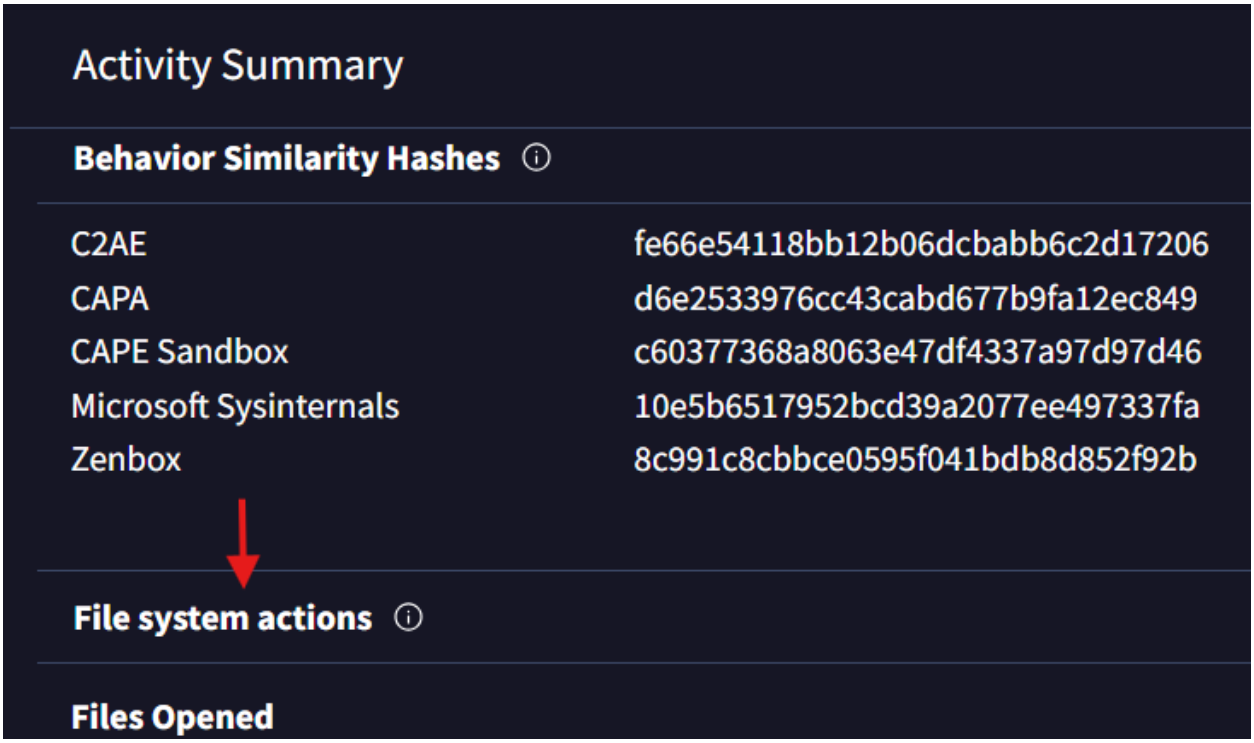
What specific filetype is enumerated by goteem.exe for encryption?

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.



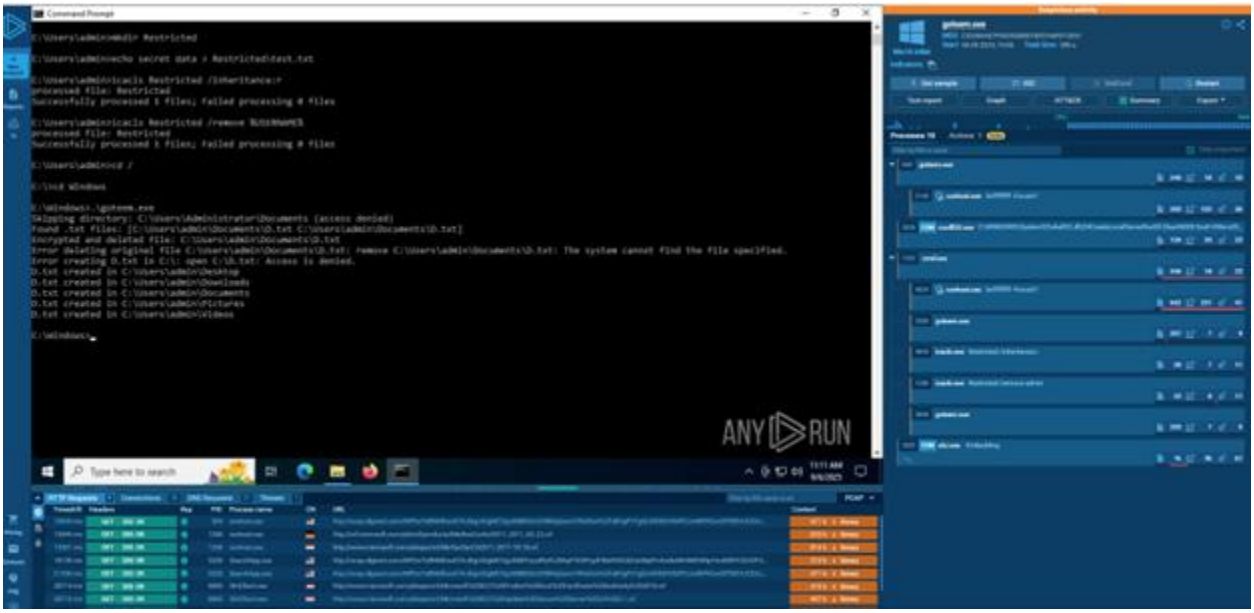


Task 8

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 cgocallbackgmakechan: invalid channel element typeunreach
able method called. linker bug?concurrent map iteration and map writegcBgMarkWorker: blackening not enabledcannot read st
ack of running goroutineruntime: blocked read on free polldescruntime: LoadLibraryExW failed; errno=runtime: GetProcAddre
ss failed; errno=runtime: sudog with non-false isSelectarg size to reflect.call more than 1GBv could not fit in traceByte
```

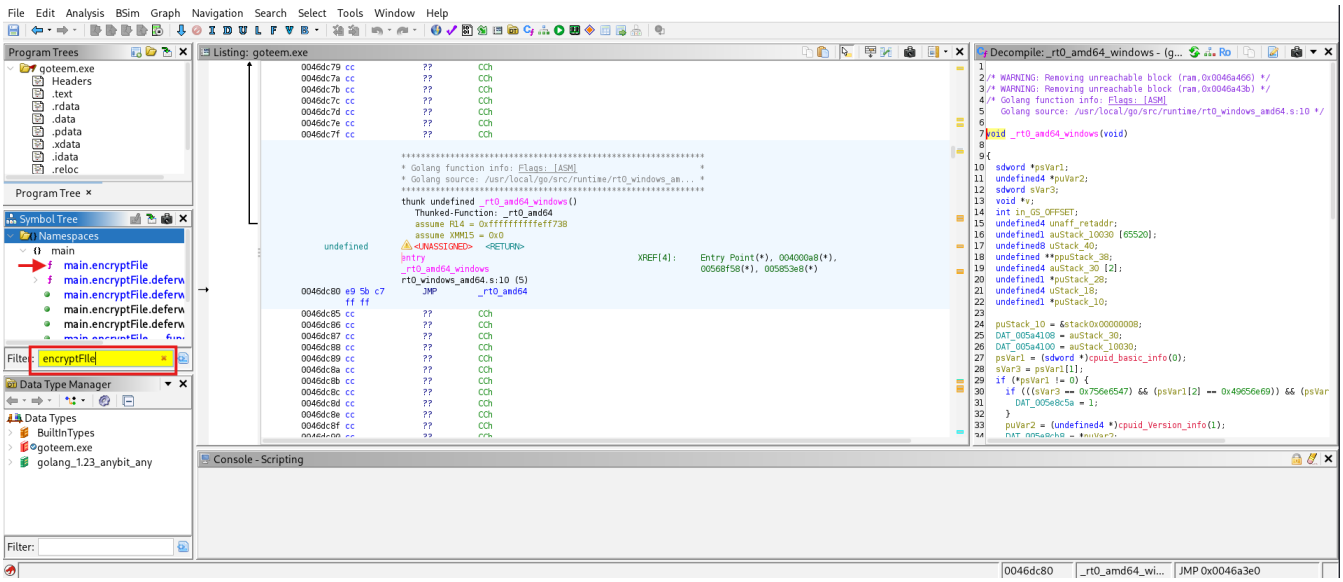
Task 9

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

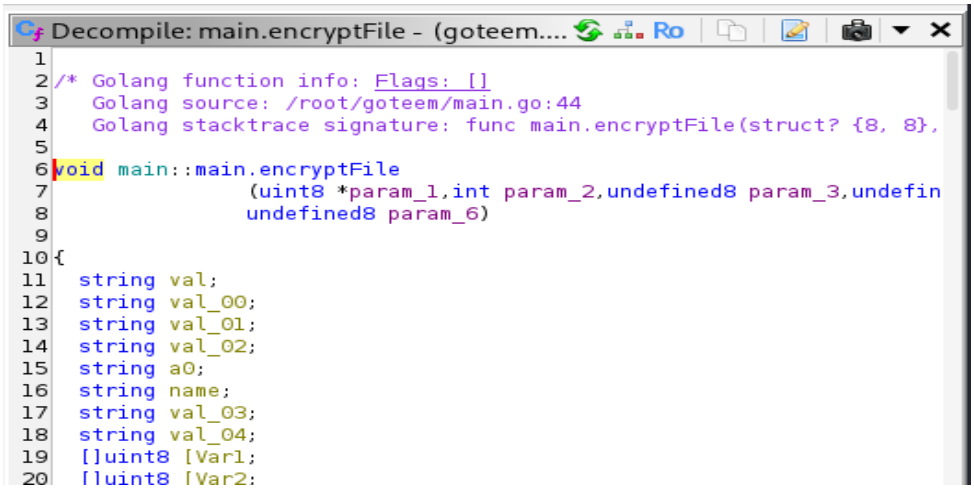
I used Ghidra for static analysis (though other decompilers could also work).

```
┌──(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.



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.



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.

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)*:

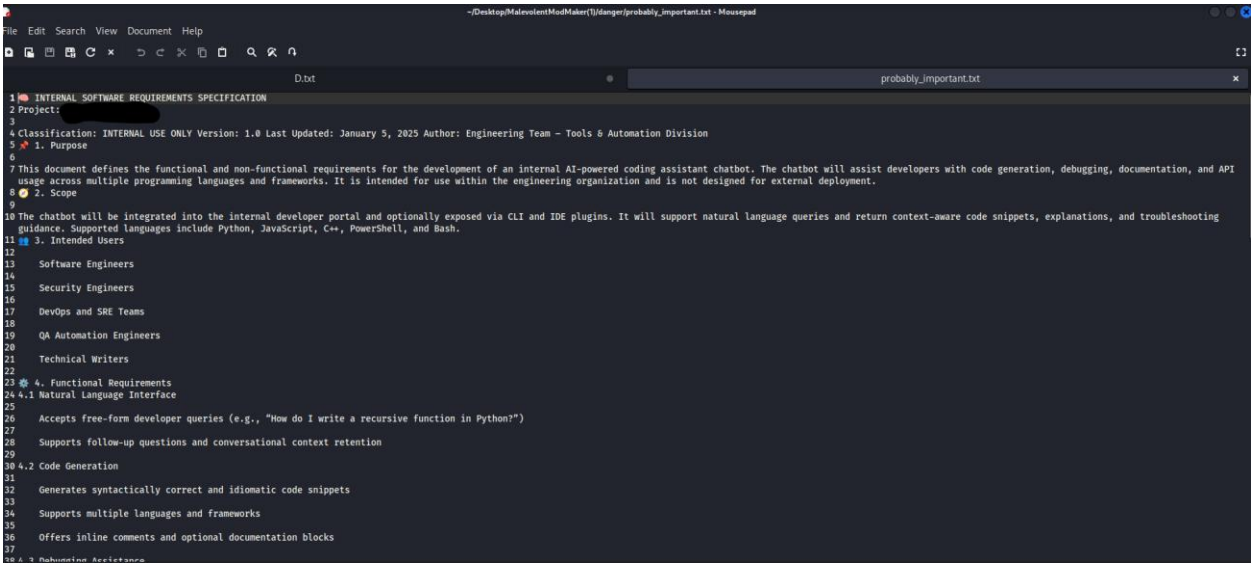
In my case I used *wine* since I’m on Kali Linux.

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.



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.