A Journey to Network Protocol Fuzzing – Dissecting Microsoft IMAP Client Protocol

fortinet.com/blog/threat-research/analyzing-microsoft-imap-client-protocol

August 3, 2022

In networking, a protocol is a set of rules that defines standard formats and processes for interpreting raw data sent by computers. Network protocols are like a common language for computers. The computers within a network may use vastly different software and hardware; however, protocols enable them to communicate with each other regardless.

Many network protocols on the Internet serve different purposes, some of which can be complex and sophisticated. Because of their inherent complexity, security vulnerabilities in network applications are inevitable. Security holes in network applications often yield a more significant security impact as compared to other attack vectors, as adversaries may be able to leverage those vulnerabilities to gain remote code execution status on vulnerable computers without any user interactions. We have seen such attacks in real-life, such as the notorious WannaCry ransomware that exploited the Simple Message Block (SMB) protocol, dubbed EternalBlue, to attack Microsoft Windows computers by encrypting data and demanding ransom payments in Bitcoin cryptocurrency. To date, this malware is estimated to have affected over 200,000 computers across 150 countries.

Hardening network applications is a mission-critical task to minimize attack vectors like WannaCry. Threat researchers work to ensure that many of the most popular network applications are properly secured using tools like network protocol fuzzing. This vulnerability discovery technique sends malformed packets to the application being tested to uncover vulnerabilities in network protocol implementations. Finding and reporting these vulnerabilities, especially in commonly used applications, help lower cyber risk for everyone.

Fortinet researchers join the rest of the threat research community in helping to achieve this goal. For this blog, we document the process of auditing and fuzzing the Microsoft Internet Message Access Protocol (IMAP) Client protocol. While we did not discover any new vulnerabilities, this step-by-step guide can help others looking to add or improve fuzzing technique strategies to their arsenal of threat discovery and analysis tools.

Why IMAP client protocol?

Network applications use client and server architectures to exchange data. However, data interpretations between client and server are different even if they share the same network protocol specifications. Data interpretation is typically performed by parsers implemented in the respective components following individual specifications. Because of this, researchers have to inspect both the client and server to ensure that the parsers are correctly implemented.

Our experience shows that servers receive more attention from researchers when auditing security implementations than clients. However, it is also possible for a less secure client to contain low-hanging fruit that can be exploited. But since we have not seen many IMAP client security issues publicly reported by vendors, we decided to look into IMAP client implementations and, at the same time, gain some hands-on experience on the open source fuzzer, What The Fuzz (WTF). WTF is a distributed, code-coverage guided, customizable, cross-platform snapshot-based fuzzer designed for attacking user and or kernel-mode targets running on Microsoft Windows.

A spoiler here: there is no vulnerability disclosure in this article as we did not discover any security issues in the Microsoft IMAP client. But we do share some rabbit holes, limitations we encountered, and tips and tricks on debugging the WTF fuzzer module. We will also walk through the reversing process for one particularly interesting IMAP response input that seemed vulnerable initially but, after digging into the code, was found to be benign.

Understanding the basic IMAP protocol

IMAP client supports a wide range of commands for different IMAP operations. The client command begins an operation and expects a response from the server. Each client command is prefixed with an identifier known as “tag”. This “tag” should be unique for every command sent by client. Generally, a client command looks like this:

```
<tag> <command> <arg1 arg2 …>
```

It is important that clients must follow the syntax as per specification strictly. It is a syntax error to send a command with missing or extraneous spaces or arguments.

An IMAP connection consists of the establishment of a client/server network connection, an initial greeting from the server, and client/server interactions. These client/server interactions consist of a client command, server data, and a server completion result response.

All interactions transmitted by client and server are in the form of lines, that is, strings that end with a carriage return and line feeds.

The first thing clients need to do is to make a connection to the remote server on a specific port. In this case, we use openssl to connect to a custom IMAP server from a terminal.

```
$ openssl s_client -connect 192.168.0.2:993 –crlf
... server verifications removed for brevity...
* OK IMAP4rev1 Service Ready
```

Noticed that server status response is prefixed with “*”, called untagged response, that indicates server greeting, or server status that does not indicate the completion of a command (for example, an impending system shutdown alert).

The next command the client typically send is CAPABILITY. The CAPABILITY command allows the client to get a listing of features that the server supports. Any features that are not listed in untagged response will be treated as BAD command by the server indicated in tagged response.

```
tag1 CAPABILITY
* CAPABILITY IMAP4rev1 LITERAL+ SASL-IR CHILDREN UNSELECT MOVE IDLE NAMESPACE CONDSTORE
COMPRESS=DEFLATE X-GM-EXT-1 METADATA ID APPENDLIMIT AUTH=PLAIN
tag1 OK CAPABILITY completed
```

Status responses can be tagged or untagged. Tagged status responses indicate the completion result (OK, NO, or BAD status) of a client command, and have a prefixed tag matching the command.

The clients have to authenticate to the IMAP server before they are able to navigate the mailbox. There is some IMAP server implementation that allows access to certain mailboxes anonymously. Like the following example, our custom IMAP server allows anonymous access. However, it is note-worthy that the list of capabilities for an authenticated client is usually different from the unauthenticated client. The logged-in client typically contains more unlocked features from the IMAP server.

```
tag2 LOGIN username password
tag2 OK LOGIN completed
```

Now that the client is logged in, it can list out folders exist in the mailbox

```
tag3:LIST
* LIST (HasNoChildren) "/" INBOX
tag3 OK LIST completed
```

Here we can see the folder hierarchy that exists in the mailbox. Folders have name attributes, indicated in parentheses. Some of attributes are useful for traversing the folder hierarchy, like HasNoChildren and HasChilden. The presence of HasNoChildren attribute indicates that the mailbox has no descendant mailboxes that are accessible to the currently authenticated client.

After knowing the folder structure of the mailbox, the client can open a session on the folder using SELECT command so that messages in the mailbox can be accessed.

```
tag4 SELECT "INBOX"
* FLAGS (Seen)
* OK [PERMANENTFLAGS (\*)] Flags permitted
* 1 EXISTS
* OK [UIDNEXT 7] Predicted next UID
* OK [UIDVALIDITY 1] UIDs valid
tag4 OK [READ-WRITE] SELECT completed
```

When a selected state is returned, the server must send the above untagged data to the client before returning an OK to the client. If the selected state is established successfully, the client is said to be in the selected state and it can search and download messages from the mailbox.

```
tag5 SEARCH ALL
* SEARCH 1
tag5 OK SEARCH completed
```

The SEARCH command searches the mailbox for messages that match the given searching criteria. The searching criteria consist of one or more search keys. It can support a more comprehensive search criteria such as finding messages that have a header with the specified field-name and that contains the specified string in the text of the header. The above example shows the simplest form of SEARCH command that will search all the available messages in the server. The untagged response indicates that there is one message available in the custom IMAP server.

Some clients provides preview of the email message. This can be done by using FETCH command to download the message header only. While UID FETCH command will download the whole email message and store it locally in the client application.

```
tag6 UID FETCH 6 (RFC822.HEADER BODY.PEEK[1])
* 1 FETCH (RFC822.HEADER {194}
From: contact@example.org
To: contact@example.org
Subject: A little message, just for you
Date: Wed, 22 Mar 2022 14:31:59 +0000
Message-ID: <0000000@localhost/>
Content-Type: text/plain


 BODY[1] {11}
Hi there :) UID 6)
tag6 OK UID FETCH completed
```

Figure 1: IMAP client-server's state and flow diagram. Adapted from IETF Internet Message Access Protocol (IMAP) - Version 4rev2

The unpaved road to the IMAP client fuzzer

A harness is required to drive the fuzzing operation along with the primary fuzzer program in modern fuzzing. While this is not mandatory for the WTF fuzzer, a dedicated fuzzer module is required. If you have experience with AFL/WinAFL, a lot of time will be spent on writing an effective harness program, but you will spend most of your time developing and troubleshooting the WTF fuzzer module. Under the hood, the WTF fuzzer acts as an emulator that programmatically emulates codes within a memory dump driven by the fuzzer module. Basically, the core of a fuzzer module consists of a function breakpoint and breakpoint handlers. These breakpoint handlers are comprised of logics that serve different purposes, such as intercepting and modifying input data consumed by the target and replicating functions such as I/O operations, registry operations, and thread scheduling. The project's repository provides a comprehensive guideline for the fuzzer development process.

As a starting point, you have to identify the target component you want to fuzz and dump a snapshot of a virtual image to be consumed by the fuzzer module. According to the documentation from the project's repository, this snapshot image is typically taken from the entry point of the target module of interest, where parser routines consume input data. In the case of the Microsoft IMAP client, InternetMail.dll is the target component that implements both the IMAP and POP3 client protocol. This DLL module is hosted by the Windows host process for services, also known as svchost.exe.

Windows Mail is the front-end user interface (UI) interacting with this module, which means the UI enables a user to set up an IMAP account and download email messages from an email server. While writing our IMAP client fuzzer module, we encountered many obstacles, some of which are, luckily, partially documented in the project's issue tracker. While the majority of the hindrances are pretty specific to whatever target you are working on, we thought it might be helpful to document these challenges and our workarounds.

Preparation for IMAP client fuzzer module development

Preparation is the key to success. This is especially true when using a new fuzzer. Writing a fuzzer module for the WTF fuzzer is not an easy task. This is because we are trying to emulate code from a memory dump. In the software emulation world, you cannot expect emulated code to behave the same as the code executed on a native device. Because of this, there are numerous roadblocks to resolve to make the emulation work as intended. So, it is essential to determine the proper tools to trace and debug the fuzzer module before starting.

WTF fuzzer supports two types of trace files, coverage trace log and Tenet trace file. Basically, a coverage trace log contains the traces for each of the instructions being executed by the emulator. It helps diagnose most of the fuzzer module issues. A Tenet trace file contains each of the executed instructions plus memory/stack data manipulated by each instruction. The Tenet plugin can only consume a Tenet trace file. Tenet is an excellent trace-record-and-replay IDA Pro plugin useful for offline debugging. The Tenet trace file generated by the WTF fuzzer can be replayed through IDA Pro. From there, it allows users to explore executed code and even analyze the data read/write into memory/stack, making debugging and troubleshooting the fuzzer module much easier.

However, the caveat is that the plugin takes a very long time to process a recorded trace file if it is too large. For example, a trace file a few gigabytes in size can easily occupy most of the host memory, which could fail to replay the traces through IDA Pro. As a workaround, we introduce a “--trace-starting-address” command line parameter to the WTF fuzzer so the fuzzer will begin its tracer only when it hits a specified address. This newly introduced command line parameter significantly reduced the trace files' size. However, the results of this filtering mechanism do not turn out to be very successful in some cases. We sometimes still get a large trace file because the starting address in the function of interest is not unique. For example, the function could be triggered in multiple locations that are non-deterministic, causing the tracer to fire unexpectedly, which defeats our objective.

Figure 2: Simple filtering for Tenet traces

After some experiments, we found the Time-Travel-Debugging (TTD) feature from WinDbg Preview that can serve the same purpose for offline debugging. WinDbg Preview will attach a running process and inject a TTD proprietary tracer DLL into the target process. The injected tracer DLL is responsible for capturing the runtime execution of the target process and keeping the executed code in the trace file stored in the physical disk. To simulate this process, we created a simple IMAP server that reads IMAP packets defined in JSON format and sends the packets to the connected client, Windows Mail, when the IMAP connection is established. At the same time, WinDbg Preview is attached to the Windows host process for service to record the code execution. The catch for this approach is you can only manually generate one execution trace at a time. However, TTD is still a helpful feature that complements the offline debugging experience.

Figure 3: Alternative approach to generating code execution traces for the target executable

Another use case leverages differential debugging techniques by comparing the traces generated by TTD and Tenet to deeply troubleshoot more problematic issues from your fuzzer module. Nevertheless, Tenet is still the first option for trace file generation to debug more complex issues encountered during fuzzer module development.

In the rest of this article, we will share some tips and tricks on pinpointing some of the more apparent issues directly from the coverage trace log instead of using a Tenet trace file. This can hopefully save you time on fuzzer module development.

Developing an IMAP client fuzzer module

The WTF fuzzer module works on top of the WTF framework. Each fuzzer module has to implement callback functions registered by the WTF framework and later triggered from the WTF executable. We will not cover the development details here since the project’s repository includes step-by-step guidelines.

IMAP includes operations for creating, deleting, and renaming mailboxes, checking for new messages, permanently removing messages, setting and clearing flags, and the selective fetching of message attributes and texts. As a result, implementing a comprehensive mutation strategy for the IMAP protocol could be time-killing. In our case, we only focused on specific IMAP commands that Windows Mail uses to interact with an IMAP server. First, we attached the WinDbg Preview debugger to the target process to generate execution traces for Windows Mail interacting with a real-world IMAP server (Gmail) to collect the typical commands in IMAP transactions. Listing 1 shows the debugger's output, consisting of IMAP commands sent to the Gmail server by a Windows Mail client.

```
0:000> bp  InternetMail!ImapCommunicationManager::_IssueCommandRaw+0x2b ".printf \"_SendText: %ma\\n\",
rdx;gc"
```

```
breakpoint 0 redefined

0:000> g

ModLoad: 00007ff9`8fcc0000 00007ff9`8fd9b000   c:\windows\system32\efswrt.dll

ModLoad: 00007ff9`a22a0000 00007ff9`a22e5000   C:\Windows\SYSTEM32\feclient.dll

ModLoad: 00000241`a5a30000 00000241`a5a34000   C:\Windows\System32\UserDataAccessRes.dll

ModLoad: 00007ff9`c9380000 00007ff9`c9401000   C:\Windows\System32\fwpuclnt.dll

_SendText: A2 NOOP

_SendText: A3 CAPABILITY

_SendText: A4 ID ("vendor" "Microsoft" "os" "Windows Mobile" "os-version" "10.0" "guid"
"45363445433632373438364639394335353946323134423434313739434334344239413838323431")

_SendText: A5 LOGIN "censoredusername" "censoredpassword"

_SendText: A6 NAMESPACE

_SendText: A7 LIST "" %

_SendText: A8 LIST "[Gmail]/" *

_SendText: A9 SELECT "INBOX" (CONDSTORE)

_SendText: A10 SEARCH UNDELETED SINCE 28-Jun-2022

_SendText: A11 FETCH 89:90 (INTERNALDATE UID FLAGS X-GM-THRID RFC822.SIZE
BODY.PEEK[HEADER.FIELDS (DATE FROM SUBJECT CONTENT-TYPE X-MS-TNEF-Correlator CONTENT-CLASS
IMPORTANCE PRIORITY X-PRIORITY THREAD-TOPIC REPLY-TO)] BODYSTRUCTURE)

ModLoad: 00007ff9`cbef0000 00007ff9`cc09e000   C:\Windows\system32\windowscodecs.dll

_SendText: A12 UID FETCH 219 (RFC822.HEADER BODY.PEEK[1.2] BODY.PEEK[2])

_SendText: A13 UID FETCH 220 (RFC822.HEADER BODY.PEEK[2])

_SendText: A14 SELECT "INBOX" (CONDSTORE)

_SendText: A15 SEARCH UNDELETED SINCE 1-Apr-2022

_SendText: A16 FETCH 58:90 (INTERNALDATE UID FLAGS X-GM-THRID RFC822.SIZE
BODY.PEEK[HEADER.FIELDS (DATE FROM SUBJECT CONTENT-TYPE X-MS-TNEF-Correlator CONTENT-CLASS
IMPORTANCE PRIORITY X-PRIORITY THREAD-TOPIC REPLY-TO)] BODYSTRUCTURE)

_SendText: A17 SELECT "INBOX" (CONDSTORE)
```

Listing 1: Debugger output IMAP commands sent by a Windows Mail client

Here, our mutation approach focuses on IMAP responses for NAMESPACE, LIST, SELECT, SEARCH, and FETCH commands. We decided to skip fuzzing the UID FETCH command because this response handler involves read/write to a message database in the local filesystem. Unfortunately, this operation cannot be trivially implemented for our case, even if WTF provides an I/O subsystem emulation framework by default. We thought this was a fair tradeoff as most of the important parsing operations, such as message header parsers, take place in the FETCH command.

An IMAP packet consists of a series of structured text messages defined by the specification here. Therefore, our IMAP packet mutation strategy also needs to be structure-aware. Inspired by the famous structure-aware mutation library libprotobuf-mutator, we used a JSON file format to store each mutated IMAP response. This JSON file will serve as an input test case to the fuzzer module. As per specification, the JSON object's critical component is the *ResponseParams*, which consists of core data that the IMAP client will interpret. Nevertheless, our mutator will focus on mutating data from *ResponseParams, ResponseStatus*, and *ResponseType*.

```
{
  "Packets": [
    {
      "Command": "NAMESPACE",
      "ResponseParams": [
        [
          "NIL NIL ((\"\" \".\"))"
        ]
      ],
      "ResponseStatus": "OK",
      "ResponseTag": "A6",
      "ResponseType": 42
    },
    {
      "Command": "LIST",
      "ResponseParams": [
        [
          "() \"/\" \"INBOX\""
        ]
      ],
      "ResponseStatus": "OK",
      "ResponseTag": "A7",
      "ResponseType": 42
    },
    {
      "Command": "LIST",
      "ResponseParams": [
    [
      "(\\All \\HasNoChildren) \"/\" \"INBOX/All Mail\""
    ]
  ],
  "ResponseStatus": "OK",
  "ResponseTag": "A8",
  "ResponseType": 42
},
{
  "Command": "SELECT",
  "ResponseParams": [
    [
      "* FLAGS (\\Seen)",
      "* OK [PERMANENTFLAGS (\\*)] Flags permitted",
      "* 1 EXISTS",
      "* 0 RECENT",
      "* OK [UIDNEXT 7] Predicted next UID",
      "* OK [UIDVALIDITY 1] UIDs valid"
    ]
  ],
  "ResponseStatus": "OK [READ-WRITE]",
  "ResponseTag": "A9",
  "ResponseType": 42
},
{
  "Command": "SEARCH",
  "ResponseParams": [
    [
      "SEARCH 1"
    ]
  ],
      "ResponseStatus": "OK",
      "ResponseTag": "A10",
      "ResponseType": 42
    },
    {
      "Command": "FETCH",
      "ResponseParams": [
        [
          "38 FETCH",
          "UID 6",
          "INTERNALDATE \"31-Mar-2022 24:59:59 +0800\"",
          "FLAGS (\\Seen ks6JUz2I )",
          "RFC822.SIZE 433313086",
          "X-GM-THRID ",
          "BODYSTRUCTURE (\"report\" \"alternative\" (\"attachment\" \"(\"filename\" \"winmail.dat\")\" \"charset\" \"(\"filename\" \"winmail.dat\")\" \"\" \"\" \"charset\" \"us-ascii\" \"charset\" \"utf-8\") i.DS. I3j7]iQxo:hxfSI \"BASE64\" 26 )",
          "BODY[HEADER.FIELDS (DATE FROM SUBJECT CONTENT-TYPE X-MS-TNEF-Correlator CONTENT-CLASS IMPORTANCE PRIORITY X-PRIORITY THREAD-TOPIC REPLY-TO)] {671}\r\nContent-Duration: 5725206\r\nContent-Type: \u0002\r\nDate: Wed, 22\n Feb 2022 10:08:--62094 \r\nFrom: sample <sample@sample.com>\r\nMIME-Version: 97.56\r\nMessage-Context: voide \r\nReceived: from DB5EUR03FT058.eop-EUR03.prod.protection.outlook.com (2603:10a6:6:2d:cafe::b0)\r\nby DB6PR07CA0011.outlook.office365.com(2603:10a6:6:2d::21)\r\nwith Microsoft SMTP Server(version = TLS1_2,cipher=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384)\r\nid 15.20.4352.8\r\nvia Frontend Transport; Wed, 14 Jul 2021 05:46:31 +0000\r\nReply-To: example/example\r\n<reply+ABFSGURWRY340282366920938463463374607431768211458OIKSMTFLHORW\r\nReturn-Path: sampleʰ\r\nSubject: Example message\r\nX-MS-TNEF-Correlator: \r\n"
        ]
      ],
      "ResponseStatus": "OK",
      "ResponseTag": "A11",
      "ResponseType": 42
    }
  ]
}
```

Listing 2: Sample IMAP response input test case

Challenge #1: Extensible Storage Engine cache clean up

Windows Mail utilizes a unified store database to keep email data, such as email addresses and messages, in a local file system. This database is located on path *%LOCALAPPDATA%\Comms\UnistoreDB\store.vol*. The Extensible Storage Engine (ESENT) builds the database using a proprietary binary format for its data structure. This binary format can be viewed using a tool like ESEDatabaseView. The benefit of using ESENT is that it has a caching mechanism to maximize high-performance access to data. It is this caching mechanism that gives us our first roadblock.

Under the hood, the cache buffer is allocated a size based on the ESENT parameter JET_paramVerPageSize that is initialized when the system service initiates UserDataService. The default cache size is 0x2000 and must be aligned with page size granularity. However, this becomes an issue in the context of the WTF fuzzer module.

The problem is that ESENT will queue a work item to purge the cache buffer when the cache buffer is full. A work item is a subroutine that a program can submit to a thread pool. The work item is executed asynchronously and will be alerted by the scheduler system based on the availability of system resources. Unfortunately, it is a complex mechanism that the WTF fuzzer cannot emulate. as a result, the fuzzer module will exit during context switch when it hits threading APIs (e.g., KERNELBASE!QueueUserWorkItem). It is a waste of CPU time to let the fuzzer go beyond the context switch. This is the reason why you should find a similar breakpoint handler in every WTF fuzzer module that looks like Figure 4:

Figure 4: Breakpoint handler to stop the fuzzer module during a context switch

When an unexpected context switch occurs, the author must understand why it happened and implement workarounds to reach the desired code path. This can be done by analyzing the coverage trace log generated by the WTF fuzzer and post-processed by 0vercl0k’s Symbolizer. An example of a coverage trace log that stops at a context switch is shown in Figure 5:

Figure 5: Example of a coverage trace log generated via Symbolizer

There are no sophisticated tricks here to analyze the coverage trace log. We simply do backtracking to locate a module or function transition (i.e., modA!funcnameX -> modB!funcnameY) to discover the cause of the context switch. Typically, we load the module file into IDA Pro to statistically study and understand the underlying code. Sometimes, it might not be sufficient to perform static code analysis, especially if the code includes virtual function calls that cannot be resolved automatically by IDA Pro. Instead, you can use TTD to resolve the virtual function call or explore the executed code.

Figure 6: Coverage trace log reveals the cause of a context switch

Figure 6 shows that *ESENT!CGPTaskManager::ErrTMPost+0xd4* calls *KERNELBASE!QueueUserWorkItem*, essentially placing an executable thread in the thread pool queue, while *ESENT!CGPTaskManager::ErrTMPost* is derived from *ESENT!VER::VERSignalCleanup*. After a deep dive in analyzing the function, with the help of TTD, we determined the purpose of *ESENT!VER::VERSignalCleanup* compares the current buffer cache size against the default cache size specified via JET_paramVerPageSize. It calls QueueUserWorkItem to execute the cache cleanup thread, *ESENT! VER::VERIRCECleanProc*, if the current cache buffer is filled up, which eventually results in a context switch. So, our challenge is to find a way to prevent the cleanup procedure from being triggered.

Our first thought was that the simplest workaround would be to increase the default cache size from 0x2000 to its maximum size of 0x10000. Technically, the configuration settings for the database engine can be adjusted using the API *JetSetSystemParameter*, as per MSDN documentation. But we cannot achieve this goal using an external program to change settings that reside in an isolated system service process space.

```
# RetAddr           : Call Site

00 00007ff9`d9773c38  : ESENT!SetConfiguration+0xce

01 00007ff9`d9773adf  : ESENT!ErrSetSystemParameter+0x10c

02 00007ff9`d9772fe1  : ESENT!JetSetSystemParameterEx+0x203
```

```
03 00007ff9`d9772e83  : ESENT!JetSetSystemParameterExA+0x65

04 00007ff9`cc099d77  : ESENT!JetSetSystemParameterA+0x53

05 00007ff9`cc099e12  : UserDataPlatformHelperUtil!SetJetSystemParameter+0x2f

06 00007ff9`e833627a  : UserDataPlatformHelperUtil!SetParametersInitOnce+0x32

07 00007ff9`e5d709b1  : ntdll!RtlRunOnceExecuteOnce+0x9a

08 00007ff9`cc09a199  : KERNELBASE!InitOnceExecuteOnce+0x21

09 00007ff9`8cb98280  : UserDataPlatformHelperUtil!SetCommsServiceJetGlobalSystemParameters+0x29  0a
00007ff9`8cb909d6  : unistore!DBManager::_InitializeJetInstance+0x134

0b 00007ff9`8cb908bd  : unistore!DBManager::InitializeDeviceVolumeImpl+0x32

0c 00007ff9`8cb786e2  : unistore!DBManager::InitializeDeviceVolume+0xd9

0d 00007ff9`8cc02743  : unistore!DBManager::GetInstance+0x8c

0e 00007ff9`8cbb49a0  : unistore!ObjectCollection::Init+0x7b

0f 00007ff9`8cbb8c5e  : unistore!CFactory::CreateObjectCollection+0x80

10 00007ff9`8cbb7a57  : unistore!CStoreManager::GetStoreCollection+0x9e

11 00007ff9`8cbb6faf  : unistore!CStoreManager::FinalConstruct+0xd7

12 00007ff9`8cbb7050  : unistore!ATL::CComObject<CStoreManager>::CreateInstance+0x5f

13 00007ff9`8cb86cca  : unistore!CStoreManager::CreateInstance+0x50

14 00007ff9`8cb86d6b  : unistore!_CreateInprocStoreManager+0x12e

15 00007ff9`8cb87373  : unistore!_CreateStoreManager+0x77

16 00007ff9`8cb8730f  : unistore!CreateStoreManagerWithToken+0x23

17 00007ff9`a9f1050e  : unistore!CreateStoreManager+0xaf

18 00007ff9`a9f117e0  : CEMAPI!GetStoreManagerCache+0x12

19 00007ff9`a9f0e407  : CEMAPI!MapiCtx::_GetSingletonInstance+0x6c

1a 00007ff9`a9f0e5fa  : CEMAPI!LogonMapi+0x97

1b 00007ff9`8c6fa92c  : CEMAPI!MAPILogonEx+0x5a

1c 00007ff9`8c6f5738  : MessagingDataModel2!CSmStore::FinalConstruct+0x64

1d 00007ff9`8c6f3794  : MessagingDataModel2!ATL::CComCreator<ATL::CComObject<CSmStore>
>::CreateInstance+0xc0

1e 00007ff9`e6e7aa6d  : MessagingDataModel2!ATL::CComClassFactory::CreateInstance+0x64

1f 00007ff9`e6e8baef  : combase!CServerContextActivator::CreateInstance+0x1fd
[onecore\com\combase\objact\actvator.cxx @ 874]

20 00007ff9`e6e79c4c  : combase!ActivationPropertiesIn::DelegateCreateInstance+0x8f
[onecore\com\combase\actprops\actprops.cxx @ 1960]
```

```
21 00007ff9`e6ee5b28  : combase!CApartmentActivator::CreateInstance+0xcc
[onecore\com\combase\object\actvator.cxx @ 2178]

22 00007ff9`e6eecaac  : combase!CProcessActivator::CCICallback+0x68 [onecore\com\combase\object\actvator.cxx
@ 1617]

23 00007ff9`e6ee1bfe  : combase!CProcessActivator::AttemptActivation+0x4c
[onecore\com\combase\object\actvator.cxx @ 1504]

24 00007ff9`e6e7a844  : combase!CProcessActivator::ActivateByContext+0x9e
[onecore\com\combase\object\actvator.cxx @ 1348]

25 00007ff9`e6e8baef  : combase!CProcessActivator::CreateInstance+0x94 [onecore\com\combase\object\actvator.cxx
@ 1248]

26 00007ff9`e6e7b604  : combase!ActivationPropertiesIn::DelegateCreateInstance+0x8f
[onecore\com\combase\actprops\actprops.cxx @ 1960]

27 00007ff9`e6e8baef  : combase!CClientContextActivator::CreateInstance+0x124
[onecore\com\combase\object\actvator.cxx @ 558]

28 00007ff9`e6e8eb83  : combase!ActivationPropertiesIn::DelegateCreateInstance+0x8f
[onecore\com\combase\actprops\actprops.cxx @ 1960]

29 00007ff9`e6e8e125  : combase!ICoCreateInstanceEx+0x8a3 [onecore\com\combase\object\object.cxx @ 1931]

2a 00007ff9`e6e8df5c  : combase!CComActivator::DoCreateInstance+0x175 [onecore\com\combase\object\immact.hxx
@ 388]

2b (Inline Function)  : combase!CoCreateInstanceEx+0xd1 [onecore\com\combase\object\actapi.cxx @ 320]

2c 00007ff9`8c76c607  : combase!CoCreateInstance+0x10c [onecore\com\combase\object\actapi.cxx @ 264]

2d 00007ff9`8c76c2e0  : MessagingDataModel2!MessagingNotificationFactory::CreateNotificationManager+0x53

2e 00007ff9`8c86be4a  : MessagingDataModel2!Messaging_StartNotification+0x30

2f 00007ff9`8c922a02  : userdataservice!DataModelService::OnStarted+0x9a

30 00007ff9`8c9223c9  : userdataservice!ServiceBase::_ServiceMainInner+0x126

31 00007ff6`8e23296e  : userdataservice!ServiceBase::ServiceMain+0x71

32 00007ff9`e71b0752  : svchost!ServiceStarter+0x5fe

33 00007ff9`e75054e0  : sechost!ScSvcctrlThreadW+0x32

34 00007ff9`e832485b  : KERNEL32!BaseThreadInitThunk+0x10

35 00000000`00000000  : ntdll!RtlUserThreadStart+0x2b
```

Listing 3: Call-stack that shows the system host service set database engine configuration settings

Looking at the call-stack in Listing 3, we then thought of tackling this issue by hijacking UserDataService and adjusting the default cache size hard coded at a specific offset in ESENT.dll before the database engine configuration setting took place. We decided to give it a shot.

Hijacking a service DLL is straightforward. We can locate the target service registry entry, which is defined under:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\UserDataSvc\Parameters
ServiceDLL= %SystemRoot%\System32\userdataservice.dll
```

When the ServiceDLL entry is adjusted to our custom service DLL file, it will become:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\UserDataSvc\Parameters
ServiceDLL=c:\userdatasvc\UserDataSvcProxy.dll
```

The custom service DLL exports two mandatory mockup functions, *ServiceMain* and *SvchostPushServiceGlobals*. After modifying the above registry entry, the system service will load the custom service DLL, which executes the mockup *ServiceMain* function. The mockup *ServiceMain* function will patch JET_paramVerPageSize at a specific offset in ESENT.dll. After patching, it will pass the execution to the initial *ServiceMain* function exported by UserDataService and continue its initial routine as usual.

The complete source code of the custom service DLL is available here.

```
# Child-SP          RetAddr           Call Site

00 000000a6`d807d3d8 00007ff9`c86209d6     unistore!DBManager::_InitializeJetInstance+0x5

01 000000a6`d807d3e0 00007ff9`c86208bd     unistore!DBManager::InitializeDeviceVolumeImpl+0x32

02 000000a6`d807d420 00007ff9`c86086e2     unistore!DBManager::InitializeDeviceVolume+0xd9

03 000000a6`d807d490 00007ff9`c8692743     unistore!DBManager::GetInstance+0x8c

04 000000a6`d807d4e0 00007ff9`c86449a0     unistore!ObjectCollection::Init+0x7b

05 000000a6`d807d710 00007ff9`c8648c5e     unistore!CFactory::CreateObjectCollection+0x80

06 000000a6`d807d770 00007ff9`c8647a57     unistore!CStoreManager::GetStoreCollection+0x9e

07 000000a6`d807d7e0 00007ff9`c8646faf     unistore!CStoreManager::FinalConstruct+0xd7

08 000000a6`d807d8e0 00007ff9`c8647050     unistore!ATL::CComObject<CStoreManager>::CreateInstance+0x5f

09 000000a6`d807d910 00007ff9`c8616cca     unistore!CStoreManager::CreateInstance+0x50

0a 000000a6`d807d950 00007ff9`c8616d6b     unistore!_CreateInprocStoreManager+0x12e

0b 000000a6`d807d990 00007ff9`c8617373     unistore!_CreateStoreManager+0x77

0c 000000a6`d807da20 00007ff9`c861730f     unistore!CreateStoreManagerWithToken+0x23

0d 000000a6`d807da70 00007ff9`c91a050e     unistore!CreateStoreManager+0xaf

0e 000000a6`d807dab0 00007ff9`c91a17e0     CEMAPI!GetStoreManagerCache+0x12

0f 000000a6`d807daf0 00007ff9`c919e407     CEMAPI!MapiCtx::_GetSingletonInstance+0x6c

10 000000a6`d807db30 00007ff9`c919e5fa     CEMAPI!LogonMapi+0x97

11 000000a6`d807db70 00007ff9`9b1aa92c     CEMAPI!MAPILogonEx+0x5a

12 000000a6`d807dbd0 00007ff9`9b1a5738     MessagingDataModel2!CSmStore::FinalConstruct+0x64

13 000000a6`d807dc20 00007ff9`9b1a3794
MessagingDataModel2!ATL::CComCreator<ATL::CComObject<CSmStore> >::CreateInstance+0xc0

14 000000a6`d807dc60 00007ff9`e4adaa6d     MessagingDataModel2!ATL::CComClassFactory::CreateInstance+0x64
```

```
15 000000a6`d807dc90 00007ff9`e4aebaef     combase!CServerContextActivator::CreateInstance+0x1fd
[onecore\com\combase\objact\actvator.cxx @ 874]

16 000000a6`d807de10 00007ff9`e4ad9c4c     combase!ActivationPropertiesIn::DelegateCreateInstance+0x8f
[onecore\com\combase\actprops\actprops.cxx @ 1960]

17 000000a6`d807dea0 00007ff9`e4b45b28     combase!CApartmentActivator::CreateInstance+0xcc
[onecore\com\combase\objact\actvator.cxx @ 2178]

18 000000a6`d807df50 00007ff9`e4b4caac     combase!CProcessActivator::CCICallback+0x68
[onecore\com\combase\objact\actvator.cxx @ 1617]

19 000000a6`d807dfa0 00007ff9`e4b41bfe     combase!CProcessActivator::AttemptActivation+0x4c
[onecore\com\combase\objact\actvator.cxx @ 1504]

1a 000000a6`d807dff0 00007ff9`e4ada844     combase!CProcessActivator::ActivateByContext+0x9e
[onecore\com\combase\objact\actvator.cxx @ 1348]

1b 000000a6`d807e080 00007ff9`e4aebaef     combase!CProcessActivator::CreateInstance+0x94
[onecore\com\combase\objact\actvator.cxx @ 1248]

1c 000000a6`d807e0d0 00007ff9`e4adb604     combase!ActivationPropertiesIn::DelegateCreateInstance+0x8f
[onecore\com\combase\actprops\actprops.cxx @ 1960]

1d 000000a6`d807e160 00007ff9`e4aebaef     combase!CClientContextActivator::CreateInstance+0x124
[onecore\com\combase\objact\actvator.cxx @ 558]

1e 000000a6`d807e200 00007ff9`e4aeeb83     combase!ActivationPropertiesIn::DelegateCreateInstance+0x8f
[onecore\com\combase\actprops\actprops.cxx @ 1960]

1f 000000a6`d807e290 00007ff9`e4aee125     combase!ICoCreateInstanceEx+0x8a3
[onecore\com\combase\objact\objact.cxx @ 1931]

20 000000a6`d807ef30 00007ff9`e4aedf5c     combase!CComActivator::DoCreateInstance+0x175
[onecore\com\combase\objact\immact.hxx @ 388]

21 (Inline Function) --------`--------     combase!CoCreateInstanceEx+0xd1 [onecore\com\combase\objact\actapi.cxx @
320]

22 000000a6`d807f070 00007ff9`9b21c607     combase!CoCreateInstance+0x10c
[onecore\com\combase\objact\actapi.cxx @ 264]

23 000000a6`d807f110 00007ff9`9b21c2e0
MessagingDataModel2!MessagingNotificationFactory::CreateNotificationManager+0x53

24 000000a6`d807f190 00007ff9`a458be4a     MessagingDataModel2!Messaging_StartNotification+0x30

25 000000a6`d807f1d0 00007ff9`a4642a02     UserDataService!DataModelService::OnStarted+0x9a

26 000000a6`d807f2f0 00007ff9`a46423c9     UserDataService!ServiceBase::_ServiceMainInner+0x126

27 000000a6`d807f320 00007ff9`9b340ca7     UserDataService!ServiceBase::ServiceMain+0x71

28 000000a6`d807f350 00007ff9`9b333666     userdatasvcproxy!SvchostPushServiceGlobals+0xcaf2

29 000000a6`d807f358 000000a6`d807f3e8     userdatasvcproxy!ServiceMain
```

Listing 4: Call-stack that shows the custom service DLL hijacking UserDataService

After everything was set, we ran the fuzzer module against the new snapshot image with the custom service DLL loaded that should adjust the cache size to 0x10000. But unfortunately, it still hits= the cleanup procedure. So, we needed to figure out another workaround.

We looked into the *ESENT!VER::VERSignalCleanup* but realizes that the function does not return a value to the caller function, which made us believe that the function routine does not care whether this cleanup procedure has been executed successfully or not. On top of that, it does not seem to keep track of any global state or event that would potentially cause unexpected behavior in ESENT. Considering all this, we decided to skip this cleanup procedure by simply setting a breakpoint to simulate this function by returning to the caller immediately when the breakpoint is hit, as shown in Figure 7:

Figure 7: Skipping ESENT!VER::VERSignalCleanup to avoid context switch

And voila! Our fuzzer module can execute beyond the cleanup procedure without hitting a context switch! However, the caveat is that this could potentially grow memory usage substantially inside the snapshot image. But this should not pose any potential issues to us, as the snapshot image will revert to its original state once a fuzzing iteration is completed. In other words, the dangling cache buffers are negligible.

Challenge #2 Loading an unloaded DLL and executing paged out memory

If you are familiar with software emulation, you understand that making an emulator behave like a native machine is impossible. The same thing applies to the WTF fuzzer. We need to find workarounds when such limitations arise. But some workarounds could be straightforward depending on the complexity of the limitations face, with some workarounds as simple as tweaking the snapshot image.

The next issue we encountered is that a context switch happens when WTF attempts to load an unloaded DLL file from the file system. Again, we determined the root cause of the issue by analyzing the coverage trace log and some code snippets, shown in Figure 8. From the coverage trace log, we can tell that the *CoCreateInstance* API is called from *MCCSEngineShared!Decode2047Header+0xfe*. This COM API is responsible for loading a COM object specified in a class ID, which in this case is CLSID_CMultiLanguage. This class ID corresponds to C:\WINDOWS\SYSTEM32\mlang.dll.

Figure 8: Diagnosis of loading an unloaded DLL file

With this information, we manually injected the COM object DLL into the target process, dumped the image as a new snapshot, and tested it. As a result, it went beyond *MCCSEngineShared!Decode2047Header*, which is good, but we faced another issue.

Figure 9: Yet another context switch due to a memory access error

Upon looking into the coverage trace log in Figure 9, we realized an unusual code execution transition from user-mode *exsmime!CMimeReader::FindBoundary* to kernel-mode *nt!MiUserFault* happens. Our experience indicates that the emulator may have hit a reserved memory address or memory address swapped out to a page file. This is a common Windows memory management mechanism to keep infrequently used memory in a page file for performance reasons. To verify that, we used WinDbg debugger to load the memory dump and examined the code specified at *exsmime!CMimeReader::FindBoundary+0x4f*, as shown in Figure 10.

Figure 10: Memory access error when calling a virtual function

It calls a virtual function from the virtual function table but the destination of the virtual function, *exsmime!CHdrContentType::value*, is determined via TTD snapshot, as shown in Figure 11:

Figure 11: Using TTD to determine the destination address of the virtual function

To work around this memory access issue, we ran the lockmem utility, which ensures that every available memory region of a specified process will persist in memory so it will not be written to a page file, which incurs page fault on access. For the best result, performing a complete memory dump is always recommended to avoid other unforeseen memory access issues. This tip is especially useful when you are fuzzing a kernel-mode component.

Challenge #3 Registry hooks

Windows registry is a hierarchical database that stores low-level settings for the Windows operating system and applications. This database keeps the information of registry hives in the file system. In other words, a registry operation involves I/O operations to some extent. Neither of these operations is supported by an emulator, so we need to replicate these functionalities.

At the time of writing, WTF provides a fshook subsystem to replicate I/O operations but not registry hook (reghook thereafter). Apparently, we cannot reuse fshook for reghook because they are different APIs, but we can adapt some of the implementations from fshook into reghook. For example, we can reuse the pseudo-handle algorithm in fshook and RegHandleTable_t class. The critical difference between fshook and reghook is how the intended contents (i.e., file contents for I/O operations and registry data for registry operations) are emulated. For reghook, if a registry operation is going to open a new handle, for example, the RegOpenKey API is called to open a handle to a particular registry key. Its corresponding hook handler redirects the API call to the native machine. In other words, the native device will attempt to open the registry key using the native API and return the handle if the registry key exists. The opened handle is valid for the native machine but not the guest machine that is the memory dump. So, a pseudo-handle should be generated and mapped to the native handle. The current implementation of reghook can be found here.

To reiterate, the current reghook implementation is incomplete and not being tested comprehensively for other targets. But it should be reasonably straightforward to expand the existing reghook to support other registry APIs.

The curious case of RFC822.SIZE

After deploying and distributing the fuzzer module, we start to harvest interesting input gathered by the fuzzers. From there, we begin to generate code traces and load those up in IDA Pro with the Lighthouse plugin for further analysis.

We first focus on reverse engineering the InternetMail.dll to find the code that manipulates the mutated input—particularly the *ResponseParams* the fuzzer feeds into the target. One interesting *ResponseParams* in the FETCH response, RFC822.SIZE, immediately caught our attention. Per the available documentation, RFC822.SIZE is one of the FETCH command's attributes that represents the message's size. Simply put, it tells the email client the size of the entire email message that arrived in the client, including email headers, contents, and attachments.

Interestingly, from the code snippet in Listing 5, the code sanitization for this value is pretty simple – merely ensuring the size of the message is not 4 gigabytes (4294967295 in base 10 or 0xFFFFFFFF in 32-bit hexadecimal). It yields an error when it does.

```
else if (!_strnicmp("RFC822.SIZE", strItem, 0xBui64))

{

  stdstring_startItemData = this->stdstring_startItemData;

  this->rfc822_size = 0;

  v11 = strtoul(stdstring_startItemData, 0i64, 10); // **(1)**

  this->rfc822_size = v11;

  if (v11 == 0xFFFFFFFF) // **(2)**

  {

        v7 = 37;

        hr = INTSAFE_E_ARITHMETIC_OVERFLOW;
  LABEL_17:

        Log_HREvent(hr, 0, "onecoreuap\\base\\mailcontactscalendarsync\\engines\\internetmail\\imap\\fetch.cpp", v7);

  }

}
```

Listing 5: Fetching RFC822.SIZE and saving it in a data structure

In (1), a zero value is returned if no valid conversion could be performed by *strtoul*. But, it seems that the code sanitization in (2) is not meaningful as a value such as 4294967294 (0xFFFFFFFE in 32-bit hexadecimal) could bypass the check and pose an arithmetic overflow if this value is going to be used in some arithmetic operation somewhere in the code. After digging into the code, we only found a single function that manipulates this value. Not surprisingly, we see the same code sanitization here.

```
__int64 __fastcall HeaderParser::_PostNewMessageCreation(HeaderParser* this, struct IMessage* lppMessage)

{

      rfc822_size = this->rfc822_size;

      if ((_DWORD)rfc822_size != 0xFFFFFFFF)

      {

            v1 = (void*)*((_QWORD*)this->pImapSyncContext + 27); // **(A)**

            if (rfc822_size <= 0xA000)

            {

                  if ((unsigned int)rfc822_size <= 0x5000ui64)

                  {

                        if ((unsigned int)rfc822_size <= 0x2800ui64)

                        {

                              offset = 0x48i64;

                              if ((unsigned int)rfc822_size > 0x1400ui64)

                                    offset = 0x50i64;

                        }

                        else

                        {

                              offset = 0x58i64;

                        }

                  }

                  else

                  {
```

```
                offset = 0x60i64;
            }
        }
        else
        {
            offset = 0x68i64;
        }
        _InterlockedExchangeAdd64((volatile signed __int64*)((char*)v1 + offset), 1ui64); // **(B)**
        _InterlockedExchangeAdd64((volatile signed __int64*)v1 + 8, (unsigned int)rfc822_size);
    }    // **(C)**
```

Listing 6: HeaderParser::_PostNewMessageCreation manipulates RFC822.SIZE

In (A), the *v1* pointer is retrieved from *pImapSyncContext*, indicating that the unknown pointer might be related to some data structure that keeps some synchronization states. Looking further at the code, we see two arithmetic operations in (B) and (C). For (B), an incremental operation takes place depending on the value of RFC822.SIZE and the result of the incremented value is saved to the *v1* pointer, while the RFC822.SIZE value is aggregated in (C). This seems to deserve a deeper look.

So, we prepare an IMAP packet that consists of multiple FETCH *ResponseParams* with bogus RFC822.SIZE and then use TTD to capture the executed code.

```
{
  "Packets": [
    {
      "ResponseParams": [
        [
          "NIL NIL ((\"\\" \".\"))"
        ]
      ],
      "ResponseType": 42,
      "Command": "NAMESPACE",
      "ResponseStatus": "OK",
      "ResponseTag": "A6"
    },
    {
      "ResponseParams": [
        [
            "() \"/\" \"INBOX\""
        ]
    ],
    "ResponseType": 42,
    "Command": "LIST",
    "ResponseStatus": "OK",
    "ResponseTag": "A7"
},
{
    "ResponseParams": [
        [
            ""
        ]
    ],
    "ResponseType": 42,
    "Command": "LIST",
    "ResponseStatus": "OK",
    "ResponseTag": "A8"
},
{
    "ResponseParams": [
        [
            "FLAGS (\\Seen)",
            "OK [PERMANENTFLAGS (\\*)] Flags permitted",
            "2 EXISTS",
            "0 RECENT",
            "OK [UIDNEXT 3] Predicted next UID",
            "OK [UIDVALIDITY 1] UIDs valid"
        ]
    ],
            "ResponseType": 42,

            "Command": "SELECT",

            "ResponseStatus": "OK [READ-WRITE]",

            "ResponseTag": "A9"

        },

        {

            "ResponseParams": [

                [

                    "SEARCH 1 2"

                ]

            ],

            "ResponseType": 42,

            "Command": "SEARCH",

            "ResponseStatus": "OK",

            "ResponseTag": "A10"

        },

        {

            "ResponseParams": [

                [

                    "1 FETCH",

                    "UID 1",

                    "INTERNALDATE \"31-Mar-2022 24:59:59 +0800\"",

                    "FLAGS (\\Seen)",

                    "RFC822.SIZE 4294967294",

                    "X-GM-THRID NIL",

                    "BODYSTRUCTURE (\"TEXT\" \"PLAIN\" () NIL NIL NIL 11 1 NIL NIL NIL NIL)",

                    "BODY[HEADER.FIELDS (DATE FROM SUBJECT CONTENT-TYPE X-MS-TNEF-Correlator CONTENT-
CLASS IMPORTANCE PRIORITY X-PRIORITY THREAD-TOPIC REPLY-TO)]{135}\r\nFrom:
contact@example.org\r\nSubject: A little message, just for you\r\nDate: Wed, 22 Mar 2022 14:31:59 +0000\r\nContent-
Type: text/plain\r\n\r\n"

                ],

                [
```

"2 FETCH",

"UID 2",

"INTERNALDATE \"31-Mar-2022 24:59:59 +0800\"",

"FLAGS (\\Seen)",

"RFC822.SIZE 4294967294",

"X-GM-THRID NIL",

"BODYSTRUCTURE (\"TEXT\" \"PLAIN\" () NIL NIL NIL 11 1 NIL NIL NIL NIL)",

"BODY[HEADER.FIELDS (DATE FROM SUBJECT CONTENT-TYPE X-MS-TNEF-Correlator CONTENT-CLASS IMPORTANCE PRIORITY X-PRIORITY THREAD-TOPIC REPLY-TO)]{135}\r\nFrom: contact@example.org\r\nSubject: B little message, just for you\r\nDate: Wed, 22 Mar 2022 14:31:59 +0000\r\nContent-Type: text/plain\r\n\r\n"

]

],

"ResponseType": 42,

"Command": "FETCH",

"ResponseStatus": "OK",

"ResponseTag": "A11"

}

]

}

Listing 7: IMAP packet with two FETCH response parameters using bogus RFC822.SIZE

```
0:000> !tt 0

Setting position to the beginning of the trace

Setting position: 78:0

(1f58.9e0): Break instruction exception - code 80000003 (first/second chance not available)

Time Travel Position: 78:0

ntdll!NtClose+0x14:

00007fff`11a834b4 c3              ret

0:000> x internetmail!*PostNewMessageCreation*

00007fff`24253a64 InternetMail!HeaderParser::_PostNewMessageCreation (protected: long __cdecl
HeaderParser::_PostNewMessageCreation

(struct IMessage *))

0:000> bp InternetMail!HeaderParser::_PostNewMessageCreation+0x84 "r r9; dc r10+30"

0:000> g
```

```
ModLoad: 00007fff`46c00000 00007fff`46c81000   C:\Windows\System32\fwpuclnt.dll

ModLoad: 00007fff`40e20000 00007fff`40e62000   C:\Windows\system32\mlang.dll

r9=00000000fffffffe

00000251`a1f590b8  00000000 00000000 00000000 00000000  ................

00000251`a1f590c8  00000000 00000000 00000000 00000000  ................

00000251`a1f590d8  00000000 00000000 00000000 00000000  ................

00000251`a1f590e8  00000000 00000000 00000001 00000000  ................

00000251`a1f590f8  00000000 00000000 00000001 00000000  ................

00000251`a1f59108  00000000 00000000 f672a68a c667e448  .........r.H.g.

00000251`a1f59118  00000000 40200000 f673a6ba c667e449  ...... @..s.I.g.

00000251`a1f59128  00000000 00000000 00000000 00000000  ................

Time Travel Position: 106C6FF:0

InternetMail!HeaderParser::_PostNewMessageCreation+0x84:

00007fff`24253ae8 f04d0fc14a40    lock xadd qword ptr [r10+40h],r9 ds:00000251`a1f590c8=0000000000000000

0:012> g

r9=00000000fffffffe

00000251`a1f590b8  00000000 00000000 00000000 00000000  ................

00000251`a1f590c8  fffffffe 00000000 00000000 00000000  ................

00000251`a1f590d8  00000000 00000000 00000000 00000000  ................

00000251`a1f590e8  00000000 00000000 00000002 00000000  ................

00000251`a1f590f8  00000000 00000000 00000001 00000000  ................

00000251`a1f59108  00000000 00000000 f672a68a c667e448  .........r.H.g.

00000251`a1f59118  00000000 40200000 f673a6ba c667e449  ...... @..s.I.g.

00000251`a1f59128  00000000 00000000 00000000 00000000  ................

Time Travel Position: 1346091:0

InternetMail!HeaderParser::_PostNewMessageCreation+0x84:

00007fff`24253ae8 f04d0fc14a40    lock xadd qword ptr [r10+40h],r9 ds:00000251`a1f590c8=00000000fffffffe

0:012> p

Time Travel Position: 1346091:1

InternetMail!HeaderParser::_PostNewMessageCreation+0x8a:

00007fff`24253aee 488364245000    and     qword ptr [rsp+50h],0 ss:000000d2`e71fe3b0=0000000000000000

0:012> dc r10+30

00000251`a1f590b8  00000000 00000000 00000000 00000000  ................

00000251`a1f590c8  fffffffc 00000001 00000000 00000000  ................

00000251`a1f590d8  00000000 00000000 00000000 00000000  ................
```

```
00000251`a1f590e8  00000000 00000000 00000002 00000000  ................

00000251`a1f590f8  00000000 00000000 00000001 00000000  ................

00000251`a1f59108  00000000 00000000 f672a68a c667e448  ..........r.H.g.

00000251`a1f59118  00000000 40200000 f673a6ba c667e449  ...... @..s.I.g.

00000251`a1f59128  00000000 00000000 00000000 00000000  ................
```

Listing 8: Debugger output shows the aggregated value of RFC822.SIZE in v1 pointer

The highlighted area in Listing 8 clearly shows that the aggregated value has overflowed the adjacent field in the *v1* pointer. But we are unsure if the overwritten field poses any security issue. So, we need to determine the data structure fields for this raw memory. We use TTD.Utility.GetHeapAddress to reveal the starting heap address and the location where this heap address is allocated and initialized.

```
0:012> dx -g @$cursession.TTD.Utility.GetHeapAddress(0x00000251`a1f590b8)

==========================================================================================================


=                            = Action   = Heap             = Address          = Size     = Flags  = (+)
TimeStart = (+) TimeEnd  =

==========================================================================================================


= [0x1151e] : [object Object] - Alloc    - 0x251a2120000    - 0x251a1f58f10    - 0x1b8    - 0x0    -
D9B98:EC      - D9B9C:9C     =

= [0x29b79] : [object Object] - Alloc    - 0x251a2120000    - 0x251a1f58f10    - 0x1b8    - 0x0    -
216478:EC     - 21647C:9C    =

= [0x2adb1] : [object Object] - Alloc    - 0x251a2120000    - 0x251a1f58f10    - 0x1b8    - 0x0    -
220BC6:EC     - 220BCA:9C    =

= [0x2c419] : [object Object] - Alloc    - 0x251a2120000    - 0x251a1f58f10    - 0x1b8    - 0x0    -
22C5A7:EC     - 22C5AB:9C    =

= [0x2d88a] : [object Object] - Alloc    - 0x251a2120000    - 0x251a1f58f10    - 0x1b8    - 0x0    -
236CF9:EC     - 236CFD:9C    =

= [0x2ea3d] : [object Object] - Alloc    - 0x251a2120000    - 0x251a1f58f10    - 0x1b8    - 0x0    -
23FCCF:EC     - 23FCD3:9C    =

= [0x49a30] : [object Object] - Alloc    - 0x251a2120000    - 0x251a1f58f10    - 0x1b8    - 0x0    -
3985B0:EC     - 3985B4:9C    =

= [0x4aeec] : [object Object] - Alloc    - 0x251a2120000    - 0x251a1f58f10    - 0x1b8    - 0x0    -
3A33F8:EC     - 3A33FC:9C    =

= [0x4c02b] : [object Object] - Alloc    - 0x251a2120000    - 0x251a1f58f10    - 0x1b8    - 0x0    -
3AC168:EC     - 3AC16C:9C    =

= [0x76275] : [object Object] - Alloc    - 0x251a2120000    - 0x251a1f58f60    - 0x190    - 0x0    -
5C6A57:A9     - 5C6A5B:9C    =

= [0x7b09a] : [object Object] - Alloc    - 0x251a2120000    - 0x251a1f58f60    - 0x190    - 0x0    -
602818:81     - 60281C:9C    =

= [0x7b14d] : [object Object] - Alloc    - 0x251a2120000    - 0x251a1f58f60    - 0x190    - 0x0    -
602E2E:81     - 602E32:9C    =

= [0x7b1c3] : [object Object] - Alloc    - 0x251a2120000    - 0x251a1f58f60    - 0x1a0    - 0x0    -
6031FD:58     - 603201:9C    = 0:012> !tt 6031FD:58
```

```
Setting position: 6031FD:58

ModLoad: 00007fff`40e20000 00007fff`40e62000   C:\Windows\system32\mlang.dll

(1e8c.360): Break instruction exception - code 80000003 (first/second chance not available)

Time Travel Position: 6031FD:58

ntdll!RtlAllocateHeap:

00007fff`515e89a0 48895c2408      mov     qword ptr [rsp+8],rbx ss:000000d2`e7dfee80=000000d200000001

0:005> kb

 # RetAddr               : Args to Child                                                           : Call Site

00 00007fff`5076c750     : 000000d2`00000001 00000000`00000000 000000d2`e7dff000 00000000`00000000 :
ntdll!RtlAllocateHeap

01 00007fff`36bb1d97     : 00007fff`36c14678 000000d2`e7dff008 00000251`a1f8be30 00000251`a1f59dd8 :
msvcrt!malloc+0x70

02 00007fff`36bb33da     : 00007fff`36c14678 00007fff`336b651c 00000000`00000000 00000251`a1f50001 :
SYNCUTIL!operator new+0x23

03 00007fff`36bcb229     : 00000000`00000001 000000d2`e7dff088 00007876`9ee69ed8 00000000`00000001 :
SYNCUTIL!operator new+0x12

04 00007fff`36bcb7c1     : 000000d2`00000001 00007fff`2420ecfc 00000251`a1f02f00 00000251`a1f02e10 :
SYNCUTIL!SyncStatsHelpers::_LookupAccountSyncStats+0xdd

05 00007fff`2420976d     : 00000251`a1f02f00 00000000`00000000 00000000`00000000 00000251`a1f02f00 :
SYNCUTIL!GetCurrentSyncStats+0x21

06 00007fff`243100d2     : 00000000`00000000 00000251`a1fa9950 000000d2`e7dff0a0 00000251`a1f59d50 :
InternetMail!ImapIdleExecutor::Initialize+0x3d

07 00007fff`2430c5f1     : 00000000`00000000 00000251`a1f08980 00000000`00000000 00000251`a1f59d50 :
SyncController!AccountSyncControllerAggregator::_Initialize+0x17e

08 00007fff`242eec39     : 00000251`a1f08988 00007fff`00000000 00000251`a1f08988 00000251`a1f08980 :
SyncController!AccountSyncControllerAggregator::CreateInstance+0x24d

09 00007fff`242ea861     : 00000251`00000000 00007fff`2435dfd0 00000251`a2120000 00007fff`515e8d78 :
SyncController!SyncBookkeeper::_AddSyncController+0x2dd

0a 00007fff`242c8fd4     : 000063c9`e8e3b474 00007fff`515e752d 00000251`a1f91b10 00007fff`51662d96 :
SyncController!SyncBookkeeper::CreateSyncController+0xf1

0b 00007fff`242c7b4b     : 00000251`a1f6bc88 000000d2`e7dff390 00000251`a1f08900 000000d2`e7dff498 :
SyncController!SyncActivityFactory::_CreateActivity+0x34

0c 00007fff`36c508ae     : 00007fff`36c76410 00000251`a1f73b90 00000000`00000000 00000000`00000000 :
SyncController!ProviderActivityFactory<SyncActivityFactory,&SyncActivityFactoryCLSID>::CreateDefaultActivityFo


0d 00007fff`36c50c13     : 00000251`a1f08900 00007fff`36c76410 00000251`a1f75048 00007fff`5076c750 :
aphostservice!StdJobProviderByClsId<_BootstrapSyncAccountsSNJobArgs,unsigned long>::CreateActivity+0x7e

0e 00007fff`36c2547e     : 00000000`00000000 000000d2`e7dff580 00007fff`36c76410 00000000`00000000 :
aphostservice!StdJob<_BootstrapSyncAccountsSNJobArgs,unsigned long>::ExecuteActivityStep+0x53

0f 00007fff`36c2455d     : 00000251`a1f6dae8 00000251`a3b5e4b0 00000251`aa0f9d00 00000000`7ffe0386 :
aphostservice!JobScheduler::_UpdateJobQueue+0xd7e

10 00007fff`5162952a     : 00000251`aa0f9dc8 00000000`00000000 00000251`aa0f9dc8 00000251`a1e02340 :
aphostservice!JobScheduler::s_BackgroundThreadProc+0x2ed
```

```
11 00007fff`515d6eb6     : 00000000`00000000 00000000`00000000 00000251`a1e02340 00000251`a1ecfdf0 :
ntdll!TppWorkpExecuteCallback+0x13a

12 00007fff`503254e0     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 :
ntdll!TppWorkerThread+0x686

13 00007fff`515c485b     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 :
KERNEL32!BaseThreadInitThunk+0x10

14 00000000`00000000     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 :
ntdll!RtlUserThreadStart+0x2b
```

Listing 9: GetHeapAddress output and heap allocation call-stack for v1 pointer

Based on the output of TTD.Utility.GetHeapAddress, we identify that the starting heap address of the *v1* pointer is 0x251a1f58f60 and is initialized from *SYNCUTIL!SyncStatsHelpers::_LookupAccountSyncStats*. Within this function, we realize that the *v1* pointer is passed to *SYNCUTIL!SyncStatsHelpers::_LoadSyncStats* that loads various stats into the data structure referenced by the *v1* pointer.

```
__int64 __fastcall SyncStatsHelpers::_LoadSyncStats(

    GUID* rguid,

    AccSyncStates_t* v1,

    struct SyncStatsHelpers::MUTABLE_SYNC_STATS* a3)

{

    hKey = 0i64;

    v5 = (HANDLE*)tlx::replace<HKEY__*, long (*)(HKEY__*), &long RegCloseKey(HKEY__*), 0>(&hKey);

    RegKey = SyncCreateRegKey(rguid, v5);

    if (RegKey < 0)

    {

        v9 = 186i64;

    LABEL_3:

        Log_HREvent_5((unsigned int)RegKey, 1i64, v7, v9);

        if (hKey)

            RegCloseKey(hKey);

        return (unsigned int)RegKey;

    }

    RegKey = SyncStatsHelpers::_LoadStatsCounter(

        (SyncStatesHelpers*)hKey,

        (HKEY)L"ReceivedMailCount",

        (const unsigned __int16*)&v22,

        v8);
```

```
if (RegKey < 0)
{
    v9 = 191i64;
    goto LABEL_3;
}
_InterlockedExchange64((volatile __int64*)&v1->AccReceivedMailCount, v22);
RegKey = SyncStatsHelpers::_LoadStatsCounter(
    (SyncStatesHelpers*)hKey,
    (HKEY)L"DeletedMailCount",
    (const unsigned __int16*)&v22,
    v11);
if (RegKey < 0)
{
    v9 = 192i64;
    goto LABEL_3;
}
_InterlockedExchange64((volatile __int64*)&v1->AccDeletedMailCount, v22);
RegKey = SyncStatsHelpers::_LoadStatsCounter(
    (SyncStatesHelpers*)hKey,
    (HKEY)L"UpdatedMailCount",
    (const unsigned __int16*)&v22,
    v12);
if (RegKey < 0)
{
    v9 = 193i64;
    goto LABEL_3;
}
_InterlockedExchange64((volatile __int64*)&v1->AccUpdatedMailCount, v22);
RegKey = SyncStatsHelpers::_LoadStatsCounter(
    (SyncStatesHelpers*)hKey,
    (HKEY)L"SentMailCount",
        (const unsigned __int16*)&v22,
        v13);
    if (RegKey < 0)
    {
        v9 = 194i64;
        goto LABEL_3;
    }
    _InterlockedExchange64((volatile __int64*)&v1->AccSentMailCount, v22);
    RegKey = SyncStatsHelpers::_LoadStatsCounter(
        (SyncStatesHelpers*)hKey,
        (HKEY)L"MailReceivedCountUnder5k",
        (const unsigned __int16*)&v22,
        v14);
    if (RegKey < 0)
    {
        v9 = 196i64;
        goto LABEL_3;
    }
    _InterlockedExchange64((volatile __int64*)&v1->AccMailReceivedCountUnder5k, v22);
    RegKey = SyncStatsHelpers::_LoadStatsCounter(
        (SyncStatesHelpers*)hKey,
        (HKEY)L"MailReceivedCount5kTo10k",
        (const unsigned __int16*)&v22,
        v15);
    if (RegKey < 0)
    {
        v9 = 197i64;
        goto LABEL_3;
    }
    _InterlockedExchange64((volatile __int64*)&v1->AccMailReceivedCount5kTo10k, v22);
    RegKey = SyncStatsHelpers::_LoadStatsCounter(
        (SyncStatesHelpers*)hKey,
        (HKEY)L"MailReceivedCount10kTo20k",
        (const unsigned __int16*)&v22,
        v16);
if (RegKey < 0)
{
        v9 = 198i64;
        goto LABEL_3;
}
_InterlockedExchange64((volatile __int64*)&v1->AccMailReceivedCount10kTo20k, v22);
RegKey = SyncStatsHelpers::_LoadStatsCounter(
        (SyncStatesHelpers*)hKey,
        (HKEY)L"MailReceivedCount20kTo40k",
        (const unsigned __int16*)&v22,
        v17);
if (RegKey < 0)
{
        v9 = 199i64;
        goto LABEL_3;
}
_InterlockedExchange64((volatile __int64*)&v1->AccMailReceivedCount20kTo40k, v22);
RegKey = SyncStatsHelpers::_LoadStatsCounter(
        (SyncStatesHelpers*)hKey,
        (HKEY)L"MailReceivedCountOver40k",
        (const unsigned __int16*)&v22,
        v18);
if (RegKey < 0)
{
        v9 = 200i64;
        goto LABEL_3;
}
```

```
    _InterlockedExchange64((volatile __int64*)&v1->AccMailReceivedCountOver40k, v22);
    RegKey = SyncStatsHelpers::_LoadStatsCounter(
        (SyncStatesHelpers*)hKey,
        (HKEY)L"ClientUnreadToReadCount",
        (const unsigned __int16*)&v22,
        v19);
    if (RegKey < 0)
    {
        v9 = 202i64;
        goto LABEL_3;
    }
    _InterlockedExchange64((volatile __int64*)&v1->AccClientUnreadToReadCount, v22);
    RegKey = SyncStatsHelpers::_LoadStatsCounter(
        (SyncStatesHelpers*)hKey,
        (HKEY)L"MailReceivedKB",
        (const unsigned __int16*)&v22,
        v20);
    if (RegKey < 0)
    {
        v9 = 205i64;
        goto LABEL_3;
    }
    _InterlockedExchange64((volatile __int64*)&v1->AccMailReceivedKB, v22 << 10);
    if (hKey)
        RegCloseKey(hKey);
    return 0i64;
}
```

Listing 10: Initializing v1 pointer with email account stats

At this point, we recovered some of the data structure fields for the *v1* pointer. We determined that the *v1* pointer is a synchronization stats object that keeps current session stats, aggregated stats, and account stats that can be represented using a data structure, as shown in Listing 11.

```
struct CSyncStates
```

```
{
  void *lpVtbl;
  DWORD RefCount;
  DWORD UnkCount;
  SyncStates_t SyncStates;
};
struct SyncStates_t
{
  LPCRITICAL_SECTION CritSecObj;
  QWORD qwUnknown08;
  QWORD qwUnknown10;
  QWORD qwUnknown18;
  QWORD qwUnknown20;
  AccSyncStats_t AccSyncStats;
  AggrSyncStats_t AggrSyncStats;
  SessionSyncStats_t SessSyncStats;
};
struct AccSyncStats_t
{
  QWORD StartAccountSyncStats;
  QWORD qwUnknown30;
  QWORD AccReceivedMailCount;
  QWORD AccDeletedMailCount;
  QWORD AccUpdatedMailCount;
  QWORD AccSentMailCount;
  QWORD qwUnknown58;
  QWORD qwUnknown60;
  QWORD AccMailReceivedKB;
  QWORD AccMailReceivedCountUnder5k;
  QWORD AccMailReceivedCount5kTo10k;
  QWORD AccMailReceivedCount10kTo20k;
  QWORD AccMailReceivedCount20kTo40k;
  QWORD AccMailReceivedCountOver40k;
  QWORD AccClientUnreadToReadCount;
};
struct AggrSyncStats_t
{
  QWORD StartAggrSyncStats;
  QWORD qwUnknown08;
  QWORD AggrReceivedMailCount;
  QWORD AggrDeletedMailCount;
  QWORD AggrUpdatedMailCount;
  QWORD AggrSentMailCount;
  QWORD qwUnknown30;
  QWORD qwUnknown38;
  QWORD AggrMailReceivedKB;
  QWORD AggrMailReceivedCountUnder5k;
  QWORD AggrMailReceivedCount5kTo10k;
  QWORD AggrMailReceivedCount10kTo20k;
  QWORD AggrMailReceivedCount20kTo40k;
  QWORD AggrMailReceivedCountOver40k;
  QWORD AggrClientUnreadToReadCount;
};
struct SessionSyncStats_t
{
  QWORD StartSessionSyncStats;
  QWORD qwUnknown08;
  QWORD SessReceivedMailCount;
  QWORD SessDeletedMailCount;
  QWORD SessUpdatedMailCount;
  QWORD SessSentMailCount;
  QWORD qwUnknown30;
  QWORD qwUnknown38;

  QWORD SessMailReceivedKB;

  QWORD SessMailReceivedCountUnder5k;

  QWORD SessMailReceivedCount5kTo10k;

  QWORD SessMailReceivedCount10kTo20k;

  QWORD SessMailReceivedCount20kTo40k;

  QWORD SessMailReceivedCountOver40k;

  QWORD SessClientUnreadToReadCount;

};
```

Listing 11: v1 pointer data structure

After applying the data structure defined in Listing 11 in IDA Pro and decompiling HeaderParser*::_PostNewMessageCreation* again, we will get better decompiled code, as shown in Listing 12:

```
__int64 __fastcall HeaderParser::_PostNewMessageCreation(HeaderParser* this, struct IMessage* lppMessage)

{

    rfc822_size = this->rfc822_size;

    if ((_DWORD)rfc822_size != 0xFFFFFFFF)

    {

        CurrentSyncStats = (SessionSyncStats_t*)*((_QWORD*)this->pImapSyncContext + 27);

        if (rfc822_size <= 0xA000)

        {

            if ((unsigned int)rfc822_size <= 0x5000ui64)

            {

                if ((unsigned int)rfc822_size <= 0x2800ui64)

                {

                    offset = 0x48i64;

                    if ((unsigned int)rfc822_size > 0x1400ui64)

                        offset = 0x50i64;

                }

                else

                {

                    offset = 0x58i64;
                }
            }
            else
            {
                offset = 0x60i64;
            }
        }
        else
        {
            offset = 0x68i64;
        }
        _InterlockedExchangeAdd64((volatile signed __int64*)((char*)CurrentSyncStats + offset), 1ui64);
        _InterlockedExchangeAdd64(
            (volatile signed __int64*)&CurrentSyncStats->SessMailReceivedKB,
            (unsigned int)rfc822_size);
    }
```

Listing 12: Better decompiled code for HeaderParser::_PostNewMessageCreation

So, it turns out that the field is a 64-bit field, which means the arithmetic overflow we suspected at first is not harmful. We also verified the code that manipulates *SessionSyncStats_t->SessMailReceivedKB,* but it turns out this value is eventually written to the registry key "MailReceivedKB".

```
__int64 __fastcall AggregateAccountSyncStats(
    GUID* AccountGuid,
    __int64 a2,
    struct SyncStatsHelpers::AccountStatSingletons** a3)
{
    SyncStates = 0i64;
    hr = SyncStatsHelpers::_LookupAccountSyncStats(AccountGuid, (SyncStates_t*)&SyncStates, a3);
    if (hr >= 0)
    {
        EnterCriticalSection((LPCRITICAL_SECTION)SyncStates);
        SyncStatsHelpers::MUTABLE_SYNC_STATS::AggregateSyncStats(&SyncStates->AggrSyncStates,
&SyncStates->SessSyncStates);
```

```
        SyncStatsHelpers::MUTABLE_SYNC_STATS::AggregateSyncStats(
            (AggrSyncStates_t*)&SyncStates->AccSyncStates,
            (SessionSyncStates_t*)&SyncStates->AggrSyncStates);
        hr = SyncStatsHelpers::_StoreSyncStats(AccountGuid, &SyncStates->AccSyncStates, v8);
        if (hr >= 0)
            hr = 0;
        else
            Log_HREvent_5((unsigned int)hr, 1i64, v10, 346i64);
        LeaveCriticalSection((LPCRITICAL_SECTION)SyncStates);
    }
    else
    {
        Log_HREvent_5((unsigned int)hr, 1i64, v5, 338i64);
    }
    return (unsigned int)hr;
}
__int64 __fastcall SyncStatsHelpers::_StoreSyncStats(
    GUID* rguid,
    AccSyncStates_t* SyncStates,
    const struct SyncStatsHelpers::MUTABLE_SYNC_STATS* a3)
{
    hKey = 0i64;
    v5 = (HANDLE*)tlx::replace<HKEY__*, long (*)(HKEY__*), &long RegCloseKey(HKEY__*), 0>(&hKey);
    RegKey = SyncCreateRegKey(rguid, v5);
    if (RegKey < 0)
    {
        v8 = 227i64;
    LABEL_3:
        Log_HREvent_5((unsigned int)RegKey, 1i64, v7, v8);
        if (hKey)
            RegCloseKey(hKey);
        return (unsigned int)RegKey;
    }
    v10 = _InterlockedExchangeAdd64((volatile signed __int64*)&SyncStates->AccReceivedMailCount, 0i64);
    RegKey = SetRegQWORD(hKey, 0i64, L"ReceivedMailCount", v10);
    if (RegKey < 0)
    {
        v8 = 231i64;
        goto LABEL_3;
    }
    v11 = _InterlockedExchangeAdd64((volatile signed __int64*)&SyncStates->AccDeletedMailCount, 0i64);
    RegKey = SetRegQWORD(hKey, 0i64, L"DeletedMailCount", v11);
    if (RegKey < 0)
    {
        v8 = 232i64;
        goto LABEL_3;
    }
    v12 = _InterlockedExchangeAdd64((volatile signed __int64*)&SyncStates->AccUpdatedMailCount, 0i64);
    RegKey = SetRegQWORD(hKey, 0i64, L"UpdatedMailCount", v12);
    if (RegKey < 0)
    {
        v8 = 233i64;
        goto LABEL_3;
    }
    v13 = _InterlockedExchangeAdd64((volatile signed __int64*)&SyncStates->AccSentMailCount, 0i64);
    RegKey = SetRegQWORD(hKey, 0i64, L"SentMailCount", v13);
    if (RegKey < 0)
    {
        v8 = 234i64;
        goto LABEL_3;
    }
    v14 = _InterlockedExchangeAdd64((volatile signed __int64*)&SyncStates->AccMailReceivedCountUnder5k, 0i64);
    RegKey = SetRegQWORD(hKey, 0i64, L"MailReceivedCountUnder5k", v14);

    if (RegKey < 0)

    {

        v8 = 236i64;

        goto LABEL_3;

    }

    v15 = _InterlockedExchangeAdd64((volatile signed __int64*)&SyncStates->AccMailReceivedCount5kTo10k, 0i64);

    RegKey = SetRegQWORD(hKey, 0i64, L"MailReceivedCount5kTo10k", v15);

    if (RegKey < 0)

    {

        v8 = 237i64;

        goto LABEL_3;

    }

    v16 = _InterlockedExchangeAdd64((volatile signed __int64*)&SyncStates->AccMailReceivedCount10kTo20k,
0i64);

    RegKey = SetRegQWORD(hKey, 0i64, L"MailReceivedCount10kTo20k", v16);

    if (RegKey < 0)

    {

        v8 = 238i64;

        goto LABEL_3;

    }

    v17 = _InterlockedExchangeAdd64((volatile signed __int64*)&SyncStates->AccMailReceivedCount20kTo40k,
0i64);

    RegKey = SetRegQWORD(hKey, 0i64, L"MailReceivedCount20kTo40k", v17);

    if (RegKey < 0)

    {

        v8 = 239i64;

        goto LABEL_3;

    }

    v18 = _InterlockedExchangeAdd64((volatile signed __int64*)&SyncStates->AccMailReceivedCountOver40k, 0i64);

    RegKey = SetRegQWORD(hKey, 0i64, L"MailReceivedCountOver40k", v18);

    if (RegKey < 0)
    {
        v8 = 240i64;
        goto LABEL_3;
    }
    v19 = _InterlockedExchangeAdd64((volatile signed __int64*)&SyncStates->AccClientUnreadToReadCount, 0i64);
    RegKey = SetRegQWORD(hKey, 0i64, L"ClientUnreadToReadCount", v19);
    if (RegKey < 0)
    {
        v8 = 242i64;
        goto LABEL_3;
    }
    v20 = _InterlockedExchangeAdd64((volatile signed __int64*)&SyncStates->AccMailReceivedKB, 0i64);
    RegKey = SetRegQWORD(hKey, 0i64, L"MailReceivedKB", v20 >> 10);
    if (RegKey < 0)
    {
        v8 = 246i64;
        goto LABEL_3;
    }
    if (hKey)
        RegCloseKey(hKey);
    return 0i64;
}
```

Listing 13: Session synchronization stats being pushed to the registry key

Conclusion

Our fuzzer module did not discover any vulnerabilities at the time of writing, and our attempt to find potential memory corruption issues through a manual code audit did not yield any results. However, we did learn a lot in the process of fuzzing the Microsoft IMAP client. The key takeaway of this research project is that we learned more about WTF fuzzer and gained some hands-on experience using it with a real-world target. And while there are still some challenges we did not cover in this article, this tool is definitely one of the must-have tools for security researchers for black-box fuzzing.

Learn more about Fortinet's FortiGuard Labs threat research and intelligence organization and the FortiGuard Security Subscriptions and Services portfolio.