2023 LONDON

4 - 6 October, 2023 / London, United Kingdom

LOOK OUT!! OUTLOOK'S GONNA GET YOU!

Anurag Shandilya

K7 Computing, India

anurag.shandilya@k7computing.com

ABSTRACT

Part of the *Microsoft Office* suite, *Outlook* is the default choice of email client for businesses of all sizes, for daily communications as well as typical calendar functionality. Unsurprisingly, its market share is a massive 40%, and with such a huge user base, it's a popular target for threat actors who gain initial access to business networks by compromising unsuspecting employees. Further, *Outlook*'s backend is *MS Exchange Server* which, in the recent past, has itself borne the brunt of several, varied critical vulnerability exploitation attacks, from ProxyShell, exploited by the Hafnium APT, to ProxyNotShell, exploited by the Play ransomware.

Cut to the present; it turns out that *MS Outlook* has its own share of critical vulnerabilities. As recently as February 2023, CVE-2023-21716 was patched in *Microsoft Word*'s RTF parser. This is a heap corruption vulnerability leading to remote code execution. *Outlook*'s preview pane is also susceptible to this vulnerability, increasing the chances of successful exploitation and compromise. An attacker need only trick a victim into merely previewing a crafted document attached to an email to achieve RCE. Although there is no current evidence of in-the-wild exploitation of this vulnerability, we believe the risk is significant and deserves researcher attention.

Even more recently, another zero-interaction *Outlook* vulnerability, CVE-2023-23397, was patched in March 2023. This vulnerability is associated with the way in which appointment reminders are configured, leading to UNC path access, which may ultimately lead to leakage of NTLM (New Technology LAN Manager) tokens to be relayed across the network. Note, this vulnerability has been reported to have been actively exploited in the wild since April 2022.

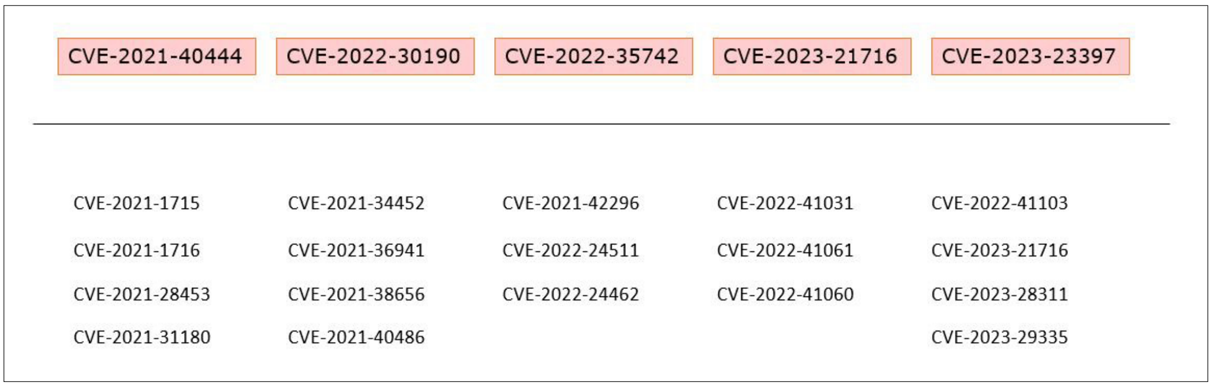
Has Pandora's Box been opened? Could there be similar, cascading *Outlook* vulnerabilities yet to be unearthed as we saw in the case of *MS Exchange Server*?

In this paper we will explain the intricate exploitation mechanisms for both CVE-2023-23397 and CVE-2023-21716. We will also analyse the TTPs threat actors have been employing to exploit *Outlook* (CVE-2023-23397, at the time of writing). The understanding thus gained will be used to project imminent in-the-wild exploitability for CVE-2023-21716, and allow us to protect against such attacks proactively.

1. VULNERABILITIES GALORE IN MS OFFICE

Microsoft (*MS*) *Outlook* is a well-known email client with a market share of around 40%. It is supported by *MS Exchange* at the backend and it allows users to configure other email providers using SMTP and POP3 as well.

With *MS* blocking execution of macros by default in *MS Office* documents, threat actors started looking for other ways to gain access to target machines. Social engineering and *One Note* files are already being utilized as initial attack vectors, but vulnerabilities in *MS Office* applications are proving to be one of the easiest methods to gain access to target systems. Our research shows that since 2021, *Microsoft* has patched no fewer than 15 vulnerabilities in *MS Word* and five in *Outlook*. Vulnerabilities in other *Windows* components are also being utilized to weaponize *MS Office* documents such as CVE-2022-30190 [1], a vulnerability in *Windows Diagnostics Tool*, and CVE-2021-40444 [2], a vulnerability in the *MS IE* rendering engine – MSHTML; both have been reported to have been exploited in the wild.



CVE-2021-40444	CVE-2022-30190	CVE-2022-35742	CVE-2023-21716	CVE-2023-23397
CVE-2021-1715	CVE-2021-34452	CVE-2021-42296	CVE-2022-41031	CVE-2022-41103
CVE-2021-1716	CVE-2021-36941	CVE-2022-24511	CVE-2022-41061	CVE-2023-21716
CVE-2021-28453	CVE-2021-38656	CVE-2022-24462	CVE-2022-41060	CVE-2023-28311
CVE-2021-31180	CVE-2021-40486			CVE-2023-29335

Figure 1: Vulnerabilities reported in Outlook and MS Word.

Figure 1 shows recent vulnerabilities reported in *Outlook* and *MS Word*. The CVEs highlighted in red are those that have been exploited in the wild, impacting *MS Office* applications directly or indirectly.

A zero-interaction vulnerability in a business email client with a high market share gives tremendous fire power to adversaries – which is proving to be the case with *Outlook*. CVE-2023-23397 [3] is one such zero-interaction vulnerability in *Outlook*. *Microsoft* disclosed that CVE-2023-23397 had been exploited as far back as April 2022. In order to exploit this vulnerability all an external threat actor needs to do is send the target a specially crafted meeting invite, which will compromise the victim's machine as soon as a reminder is triggered, irrespective of whether or not the invite has been accepted.

CVE-2023-21716 [4], on the other hand, is a vulnerability reported in *MS Word*'s RTF parser. However, thanks to the sharing of DLLs in *Windows* across applications, the preview panes in both *Outlook* and *Explorer* are susceptible to this vulnerability.

2. MAPI

The MS Outlook Messaging API (MAPI) is a framework set of APIs that allow developers to create client applications for different messaging systems, allowing maintenance of mailboxes. The framework is supported by multiple DLLs which provide an interface between front-end and back-end service providers. It provides a uniform way for multiple client applications to interact with multiple messaging systems.

MAPI architecture is shown in Figure 2. The main components are:

- Client – for users to interact with, e.g. *Outlook*
- Service providers including:
  - Message store providers, which manage storage and retrieval of messages
  - Address book providers, which are responsible for maintaining directory information
  - Transport providers, which handle message transmissions

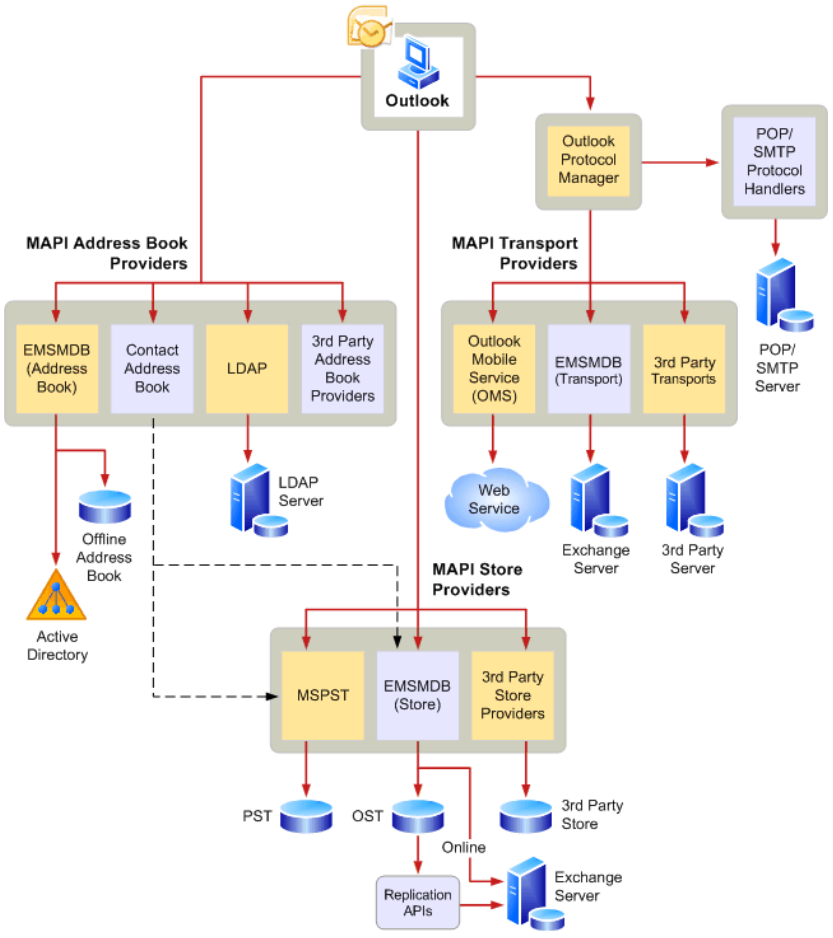


Figure 2: MAPI architecture (source: [5]).

MAPI defines various interfaces, functions and properties that clients can utilize to send and receive messages. MAPI defines object classes in interfaces such as IMAPIProp and IMAPITable. MAPI properties are attributes of these MAPI objects. Open-source tools such as MFCMAPI [6] and OutlookSpy [7] can be used to view these MAPI properties.

Transport-Neutral Encapsulation Format (TNEF) is a format that encapsulates MAPI properties into a data stream to allow messaging systems that do not support MAPI properties to transmit messages. TNEF defines attributes that are mapped to MAPI properties and store their data. TNEF and Rich Text Format (RTF) for mail (not to be confused with RTF document file format) are similar. A TNEF message contains a plain text message and a `winmail.dat` attachment that contains the original version along with MAPI properties. Clients that do not understand TNEF can choose to ignore or remove the attached DAT file. One can set the message format to HTML, Plain Text or Rich Text within *Outlook*.

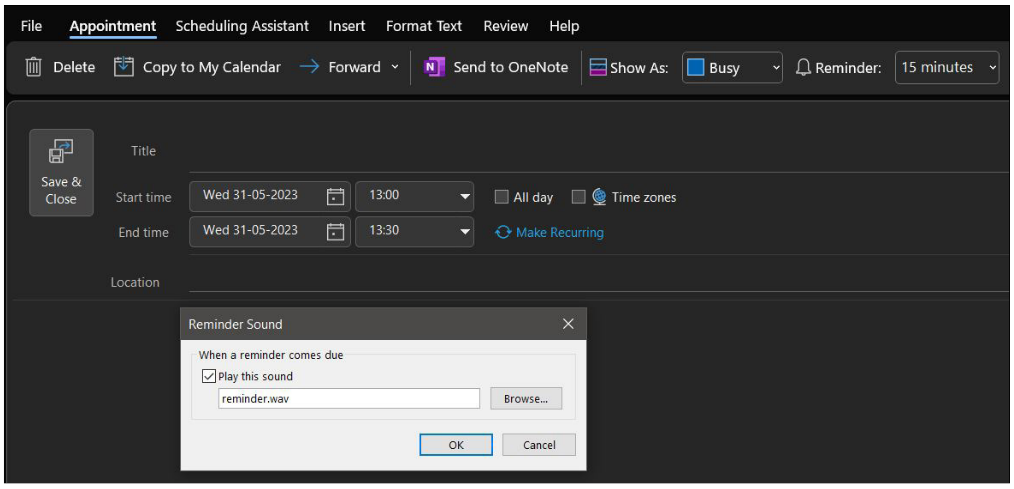
3. DEEP DIVE INTO OUR TARGET VULNERABILITIES

3.1 CVE-2023-23397

In March 2023, *Microsoft* patched CVE-2023-23397, a zero-interaction elevation of privilege (EoP) vulnerability in *Outlook*. This vulnerability allowed an adversary to send a specially crafted invite to *Outlook* users and force the system to reveal the New Technology LAN Manager (NTLM) token of the logged-in user over an unprotected network. This NTLM token can then be captured by the adversary and replayed in the network to perform other malicious activities. *Microsoft* has published evidence of in-the-wild exploitation of this vulnerability dating back to April 2022 [8].

The recipient is not even required to accept the invitation. The NTLM token sent will be of the user logged onto the system irrespective of the user logged into *Outlook*. This makes it more dangerous, since a major proportion of small and medium enterprises (SMEs) do not have user access permissions configured properly, resulting in *Windows* users with high privileges across the organization.

When creating a new appointment or task, *Outlook* allows users to update reminder settings, including the 'Reminder Sound' that should be played when a reminder comes due. As shown in Figure 3, the default file value is `remider.wav`, present in the *Office* installation location.



Show As: Busy
Reminder: 15 minutes
Start time Wed 31-05-2023 13:00
End time Wed 31-05-2023 13:30
All day ☐ Time zones ☐

Reminder Sound
When a reminder comes due
☑ Play this sound
reminder.wav

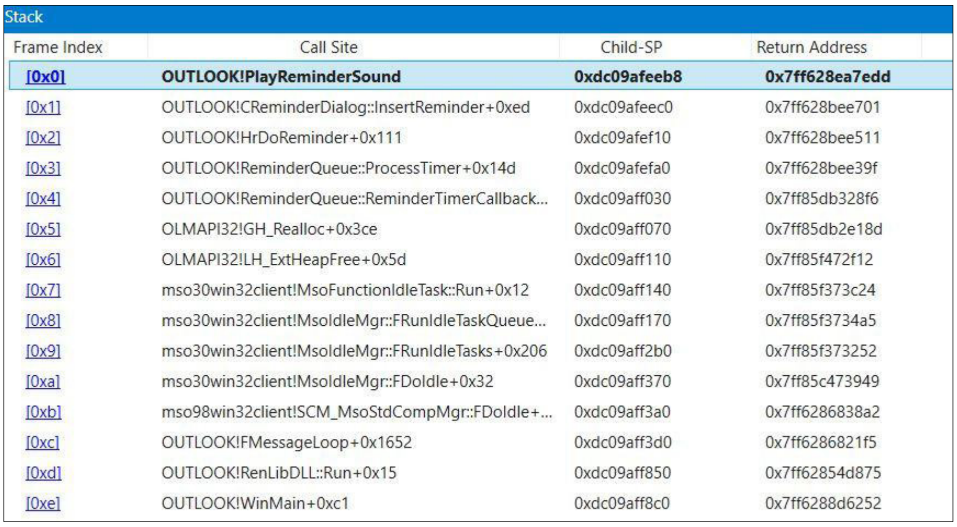
Figure 3: Outlook appointment reminder dialog.

The path for the reminder sound file that is played by default is read from

```
HKEY_USERS\S-1-5-19\AppEvents\Schemes\Apps\.Default\Notification.Reminder\.Current
```

When no alternative file value is set, the default path is read from the above location and an asynchronous thread is created to play the reminder sound using the `HrAsyncPlayReminderSound` API.

The reminder is inserted in the queue by the `CReminderDialog::InsertReminder` API, initiated by `HrDoReminder`, which ultimately calls `PlayReminderSound`, as shown in Figure 4. It is in this function that the relevant file is accessed and played.



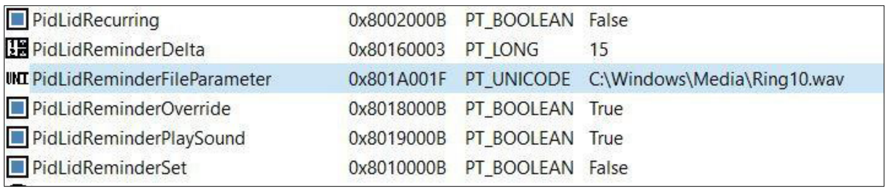
Stack

Frame Index	Call Site	Child-SP	Return Address
[0x0]	**OUTLOOK!PlayReminderSound**	**0xdc09afeeb8**	**0x7ff628ea7edd**
[0x1]	OUTLOOK!CReminderDialog::InsertReminder+0xed	0xdc09afeec0	0x7ff628bee701
[0x2]	OUTLOOK!HrDoReminder+0x111	0xdc09afef10	0x7ff628bee511
[0x3]	OUTLOOK!ReminderQueue::ProcessTimer+0x14d	0xdc09afefa0	0x7ff628bee39f
[0x4]	OUTLOOK!ReminderQueue::ReminderTimerCallback...	0xdc09aff030	0x7ff85db328f6
[0x5]	OLMAPI32!GH_Realloc+0x3ce	0xdc09aff070	0x7ff85db2e18d
[0x6]	OLMAPI32!LH_ExtHeapFree+0x5d	0xdc09aff110	0x7ff85f472f12
[0x7]	mso30win32client!MsoFunctionIdleTask::Run+0x12	0xdc09aff140	0x7ff85f373c24
[0x8]	mso30win32client!MsoIdleMgr::FRunIdleTaskQueue...	0xdc09aff170	0x7ff85f3734a5
[0x9]	mso30win32client!MsoIdleMgr::FRunIdleTasks+0x206	0xdc09aff2b0	0x7ff85f373252
[0xa]	mso30win32client!MsoIdleMgr::FDoIdle+0x32	0xdc09aff370	0x7ff85c473949
[0xb]	mso98win32client!SCM_MsoStdCompMgr::FDoIdle+...	0xdc09aff3a0	0x7ff6286838a2
[0xc]	OUTLOOK!FMessageLoop+0x1652	0xdc09aff3d0	0x7ff6286821f5
[0xd]	OUTLOOK!RenLibDLL::Run+0x15	0xdc09aff850	0x7ff62854d875
[0xe]	OUTLOOK!WinMain+0xc1	0xdc09aff8c0	0x7ff6288d6252

Figure 4: Stack trace when PlayReminderSound is called.

MS has published a Reminder Setting Protocol document [9] detailing the MAPI properties configured when a reminder is set. This set of properties remains the same irrespective of the *Outlook* item (task, appointment or meeting) the reminder is set for.

When the reminder sound file is set, the MAPI property `PidLidReminderFileParameter` points to the path to the file. If this value is not present, the property is not set and *Outlook* falls back to using the default value described earlier. Figure 5 shows this property value.



PidLidRecurring	0x8002000B	PT_BOOLEAN	False
PidLidReminderDelta	0x80160003	PT_LONG	15
PidLidReminderFileParameter	0x801A001F	PT_UNICODE	C:\Windows\Media\Ring10.wav
PidLidReminderOverride	0x8018000B	PT_BOOLEAN	True
PidLidReminderPlaySound	0x8019000B	PT_BOOLEAN	True
PidLidReminderSet	0x8010000B	PT_BOOLEAN	False

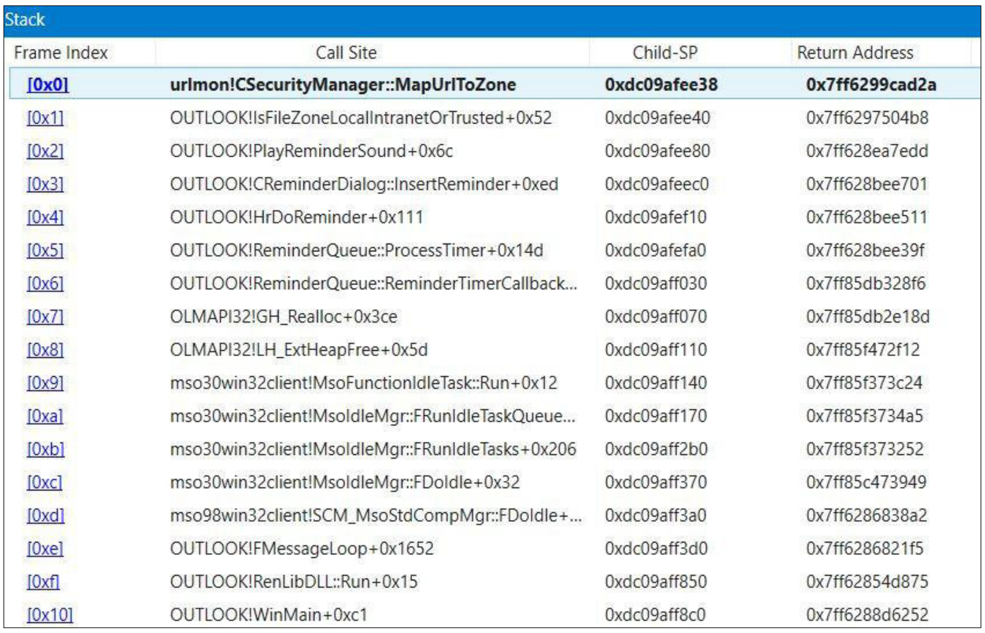
Figure 5: ReminderFileParameter MAPI property set.

Now, what if a user is tricked into passing a link to an untrusted resource instead of a local sound file path? This is where the vulnerability exists. If any user sets this property to access a file on a remote server using the Server Message Block (SMB) protocol, a file access request will be triggered to that remote server. Since SMB uses the NTLM token to authenticate a user, a request with that NTLM token will be sent to the untrusted server. This access request is sent as soon as the reminder is triggered. If the reminder is set to recur, multiple requests are sent. This attack succeeds because the file path mentioned for the reminder sound was not validated properly.

For this type of exploitation to work in practice, the following conditions must be satisfied:

1. SMB traffic must be allowed through the organization's perimeter firewall.
2. The user must be using a vulnerable version of *Outlook* client. (Note: *Outlook* clients for *iOS*, *Android* and Web (OWA) are not vulnerable to this attack.)

To fix the vulnerability, *MS* released a patch adding the API `IsFileZoneLocalIntranetorTrusted` to be called from `PlayReminderSound` to validate if the path being accessed is trusted or not, within Outlook.exe. This function, in turn, calls `MapUrlToZone` from `Urlmon.dll` (Figure 6).

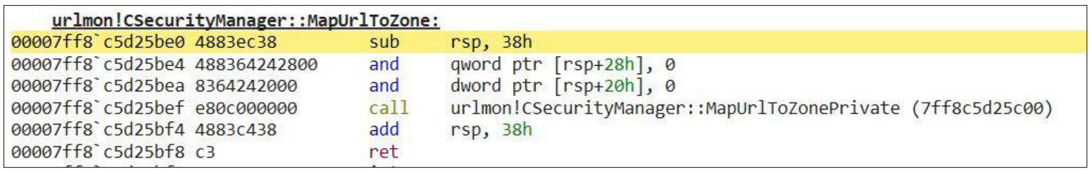


Stack

Frame Index	Call Site	Child-SP	Return Address
[0x0]	**urlmon!CSecurityManager::MapUrlToZone**	**0xdc09afee38**	**0x7ff6299cad2a**
[0x1]	OUTLOOK!IsFileZoneLocalIntranetOrTrusted+0x52	0xdc09afee40	0x7ff6297504b8
[0x2]	OUTLOOK!PlayReminderSound+0x6c	0xdc09afee80	0x7ff628ea7edd
[0x3]	OUTLOOK!CReminderDialog::InsertReminder+0xed	0xdc09afeec0	0x7ff628bee701
[0x4]	OUTLOOK!HrDoReminder+0x111	0xdc09afef10	0x7ff628bee511
[0x5]	OUTLOOK!ReminderQueue::ProcessTimer+0x14d	0xdc09afefa0	0x7ff628bee39f
[0x6]	OUTLOOK!ReminderQueue::ReminderTimerCallback...	0xdc09aff030	0x7ff85db328f6
[0x7]	OLMAPI32!GH_Realloc+0x3ce	0xdc09aff070	0x7ff85db2e18d
[0x8]	OLMAPI32!LH_ExtHeapFree+0x5d	0xdc09aff110	0x7ff85f472f12
[0x9]	mso30win32client!MsoFunctionIdleTask::Run+0x12	0xdc09aff140	0x7ff85f373c24
[0xa]	mso30win32client!MsoIdleMgr::FRunIdleTaskQueue...	0xdc09aff170	0x7ff85f3734a5
[0xb]	mso30win32client!MsoIdleMgr::FRunIdleTasks+0x206	0xdc09aff2b0	0x7ff85f373252
[0xc]	mso30win32client!MsoIdleMgr::FDoIdle+0x32	0xdc09aff370	0x7ff85c473949
[0xd]	mso98win32client!SCM_MsoStdCompMgr::FDoIdle+...	0xdc09aff3a0	0x7ff6286838a2
[0xe]	OUTLOOK!FMessageLoop+0x1652	0xdc09aff3d0	0x7ff6286821f5
[0xf]	OUTLOOK!RenLibDLL::Run+0x15	0xdc09aff850	0x7ff62854d875
[0x10]	OUTLOOK!WinMain+0xc1	0xdc09aff8c0	0x7ff6288d6252

Figure 6: Stack trace when MapUrlToZone is called.

The `MapUrlToZone` API is shown in Figure 7.



```
    urlmon!CSecurityManager::MapUrlToZone:
00007ff8`c5d25be0 4883ec38         sub     rsp, 38h
00007ff8`c5d25be4 488364242800     and     qword ptr [rsp+28h], 0
00007ff8`c5d25bea 8364242000       and     dword ptr [rsp+20h], 0
00007ff8`c5d25bef e80c000000       call    urlmon!CSecurityManager::MapUrlToZonePrivate (7ff8c5d25c00)
00007ff8`c5d25bf4 4883c438         add     rsp, 38h
00007ff8`c5d25bf8 c3               ret
```

Figure 7: MapUrlToZone function called from Urlmon.dll.

Figure 8 shows the difference in the `PlayReminderSound` function graph between the vulnerable and patched versions of Outlook.exe. The highlighted code block is where the new call is added.

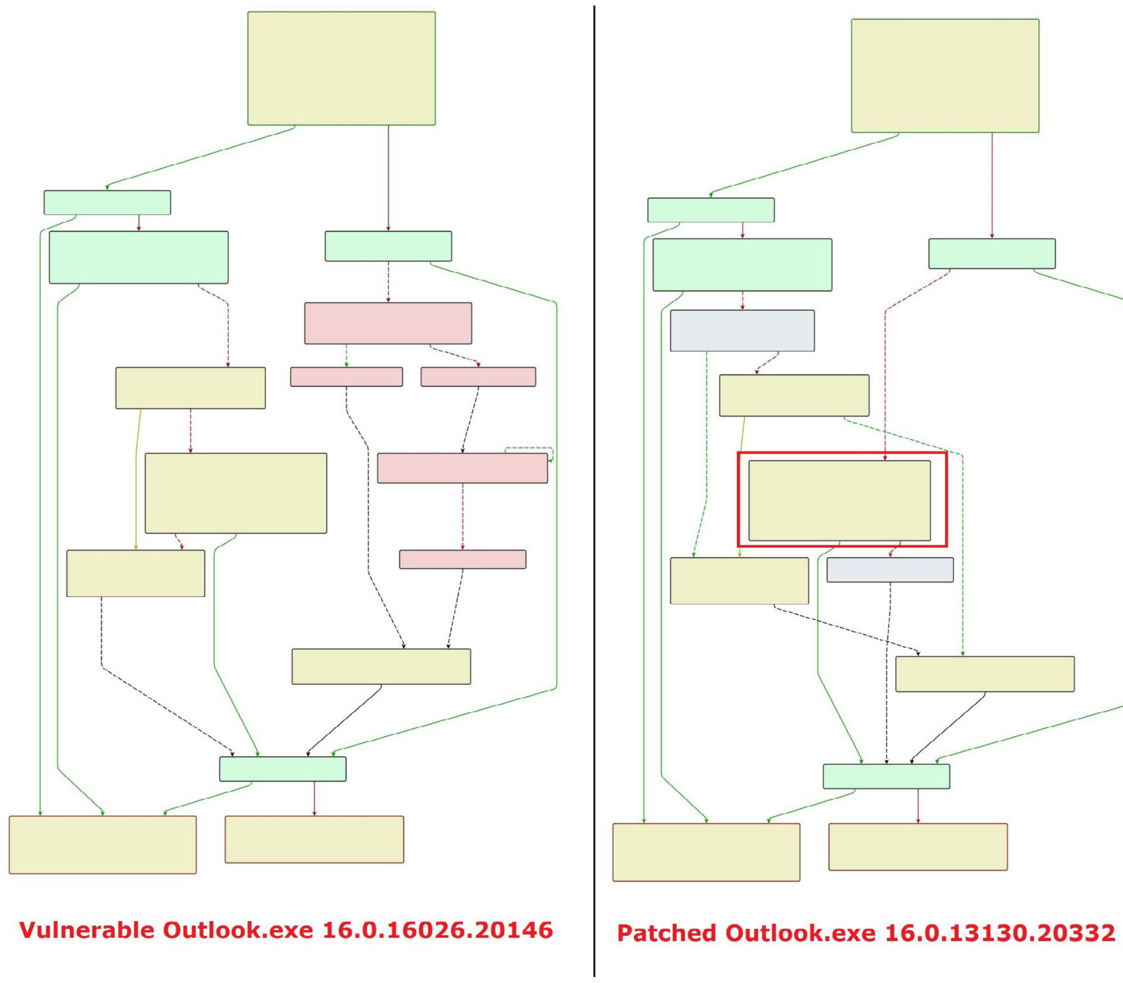
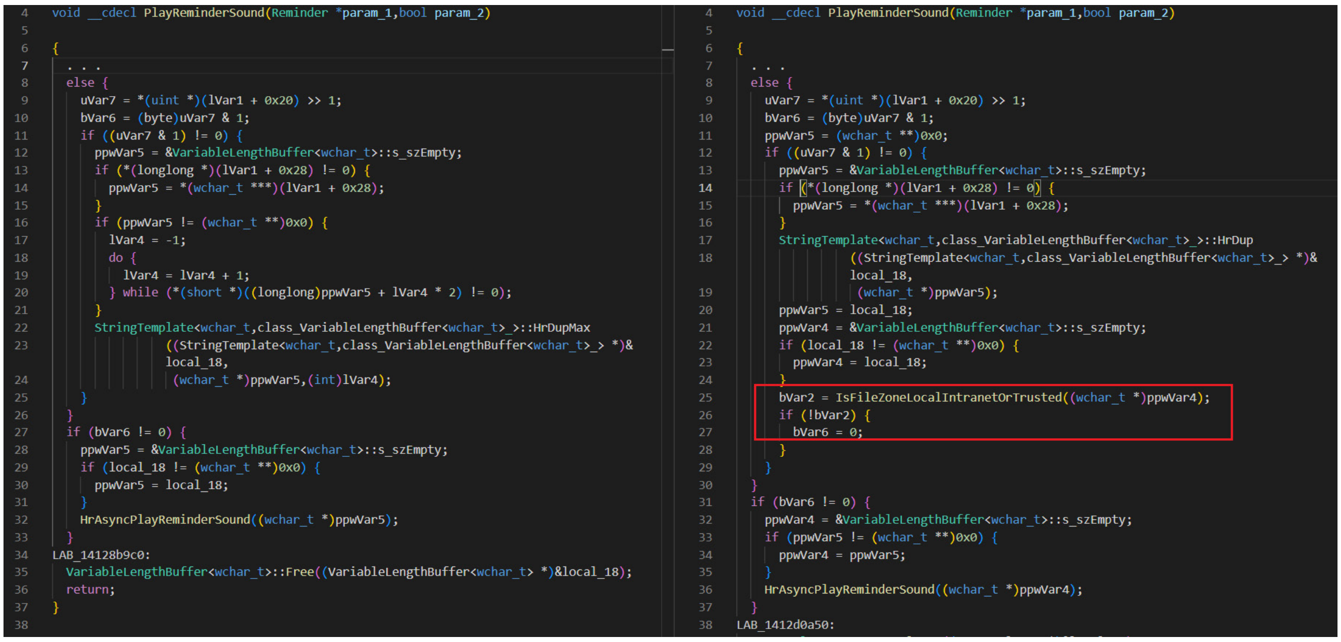

Vulnerable Outlook.exe 16.0.16026.20146
Patched Outlook.exe 16.0.13130.20332

Figure 8: Difference in function graph for PlayReminderSound.

The call to `IsFileZoneLocalIntranetorTrusted` present in the patched version of Outlook.exe is shown in Figure 9.



```
void __cdecl PlayReminderSound(Reminder *param_1,bool param_2)

{
  . . .
  else {
    uVar7 = *(uint *)(lVar1 + 0x20) >> 1;
    bVar6 = (byte)uVar7 & 1;
    if ((uVar7 & 1) != 0) {
      ppwVar5 = &VariableLengthBuffer<wchar_t>::s_szEmpty;
      if (*(longlong *)(lVar1 + 0x28) != 0) {
        ppwVar5 = *(wchar_t ***)(lVar1 + 0x28);
      }
      if (ppwVar5 != (wchar_t **)0x0) {
        lVar4 = -1;
        do {
          lVar4 = lVar4 + 1;
        } while (*(short *)((longlong)ppwVar5 + lVar4 * 2) != 0);
      }
      StringTemplate<wchar_t,class_VariableLengthBuffer<wchar_t>_>::HrDupMax
                ((StringTemplate<wchar_t,class_VariableLengthBuffer<wchar_t>_> *)&
                 local_18,
                 (wchar_t *)ppwVar5,(int)lVar4);
    }
  }
  if (bVar6 != 0) {
    ppwVar5 = &VariableLengthBuffer<wchar_t>::s_szEmpty;
    if (local_18 != (wchar_t **)0x0) {
      ppwVar5 = local_18;
    }
    HrAsyncPlayReminderSound((wchar_t *)ppwVar5);
  }
LAB_14128b9c0:
  VariableLengthBuffer<wchar_t>::Free((VariableLengthBuffer<wchar_t> *)&local_18);
  return;
}
```

```
void __cdecl PlayReminderSound(Reminder *param_1,bool param_2)

{
  . . .
  else {
    uVar7 = *(uint *)(lVar1 + 0x20) >> 1;
    bVar6 = (byte)uVar7 & 1;
    ppwVar5 = (wchar_t **)0x0;
    if ((uVar7 & 1) != 0) {
      ppwVar5 = &VariableLengthBuffer<wchar_t>::s_szEmpty;
      if ((*(longlong *)(lVar1 + 0x28) != 0)) {
        ppwVar5 = *(wchar_t ***)(lVar1 + 0x28);
      }
      StringTemplate<wchar_t,class_VariableLengthBuffer<wchar_t>_>::HrDup
                ((StringTemplate<wchar_t,class_VariableLengthBuffer<wchar_t>_> *)&
                 local_18,
                 (wchar_t *)ppwVar5);
      ppwVar5 = local_18;
      ppwVar4 = &VariableLengthBuffer<wchar_t>::s_szEmpty;
      if (local_18 != (wchar_t **)0x0) {
        ppwVar4 = local_18;
      }
      bVar2 = IsFileZoneLocalIntranetOrTrusted((wchar_t *)ppwVar4);
      if (!bVar2) {
        bVar6 = 0;
      }
    }
  }
  if (bVar6 != 0) {
    ppwVar4 = &VariableLengthBuffer<wchar_t>::s_szEmpty;
    if (ppwVar5 != (wchar_t **)0x0) {
      ppwVar4 = ppwVar5;
    }
    HrAsyncPlayReminderSound((wchar_t *)ppwVar4);
  }
LAB_1412d0a50:
```

Figure 9: PlayReminderSound source code diffing between vulnerable (left) and patched version (right).

NTLM tokens are used by multiple services in an enterprise environment for authentication. In case any service is misconfigured to accept connections from untrusted networks, or if an attacker can gain access to the organization's network, additional attacks can be mounted using, e.g. replaying, leaked NTLM tokens. In *Exchange Server*, Exchange Web Services (EWS) and Autodiscover allow NTLM token-based authentication. Several tools in the public domain allow an attacker to scan for endpoints that accept NTLM tokens and NTLM directories, making it easy to identify targets where leaked NTLM tokens can be replayed.

Along with the patch, *MS* released details of the attack campaign that had been identified exploiting this vulnerability, as well as the associated IoCs and mitigations to help organizations combat the issue. As mentioned earlier, the first known exploitation attempt was traced back to April 2022.

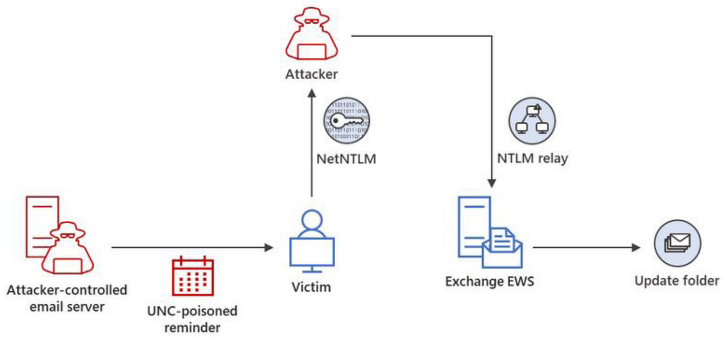


Figure 10: Attack chain observed by MS (source: [8]).

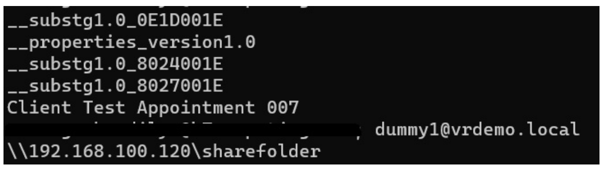
An adversary can cause the victim's machine to leak NTLM credentials and utilize those to access and modify mailbox folders properties using *Exchange* EWS (Figure 10).

MS released a PowerShell script [10] to check if any messages have the `ReminderFileParameter` property set. Figure 11 shows a snippet from the PowerShell script checking for `ReminderFileParameter Property`.

```
$searchFilterPidLidReminderFileParameterExists = New-Object Microsoft.Exchange.WebServices.Data.SearchFilter+Exists($mailInfo[
"PidLidReminderFileParameter"])
$searchFilterCollection.Add($searchFilterPidLidReminderFileParameterExists)
```

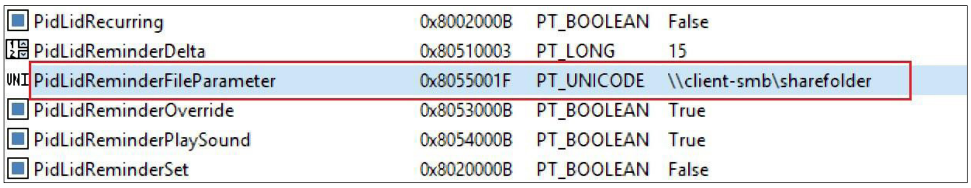
Figure 11: Code snippet from PowerShell script to check for ReminderFileParameter.

MS also shared a *VirusTotal* (*VT*) query to search for the IoCs related to the attacks. On analysing these samples, all had `ReminderFileParameter` values of the form shown in Figure 12, trying to access a Universal Naming Convention (UNC) path not validated by *Outlook*, resulting in leaking NTLM tokens to insecure and untrusted networks.



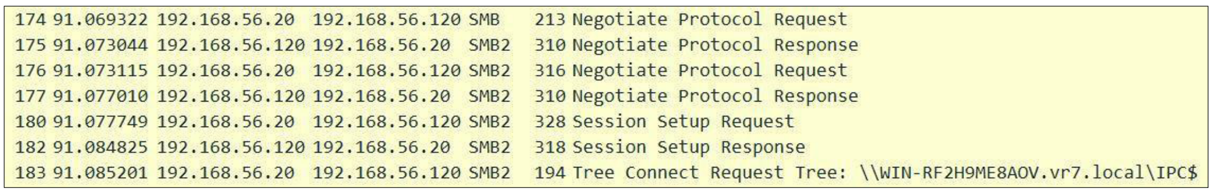
```
__substg1.0_0E1D001E
__properties_version1.0
__substg1.0_8024001E
__substg1.0_8027001E
Client Test Appointment 007
[redacted] dummy1@vrdemo.local
\\192.168.100.120\sharefolder
```

Figure 12: UNC path similar to what is present in the IoCs shared on VT.



PidLidRecurring	0x8002000B	PT_BOOLEAN	False
PidLidReminderDelta	0x80510003	PT_LONG	15
PidLidReminderFileParameter	0x8055001F	PT_UNICODE	\\client-smb\sharefolder
PidLidReminderOverride	0x8053000B	PT_BOOLEAN	True
PidLidReminderPlaySound	0x8054000B	PT_BOOLEAN	True
PidLidReminderSet	0x8020000B	PT_BOOLEAN	False

Figure 13: PidLidReminderFileParameter set in our test environment.



```
174 91.069322 192.168.56.20  192.168.56.120 SMB   213 Negotiate Protocol Request
175 91.073044 192.168.56.120 192.168.56.20  SMB2  310 Negotiate Protocol Response
176 91.073115 192.168.56.20  192.168.56.120 SMB2  316 Negotiate Protocol Request
177 91.077010 192.168.56.120 192.168.56.20  SMB2  310 Negotiate Protocol Response
180 91.077749 192.168.56.20  192.168.56.120 SMB2  328 Session Setup Request
182 91.084825 192.168.56.120 192.168.56.20  SMB2  318 Session Setup Response
183 91.085201 192.168.56.20  192.168.56.120 SMB2  194 Tree Connect Request Tree: \\WIN-RF2H9ME8AOV.vr7.local\IPC$
```

Figure 14: SMB traffic observed in our test environment.

In our test environment, we were able to add a crafted *Outlook* appointment which sets a UNC path in the reminder sound file parameter shown in Figure 13. When the reminder was triggered, an SMB connection was established with our target server, as shown in Figure 14. Multiple open-source tools are available to collect NTLM tokens and relay them to any service that accepts NTLM authentication.

Patch bypass

In May 2023, it was reported that researchers had been able to bypass the patch released by *Microsoft* for this vulnerability [11]. The bypass was assigned CVE-2023-29324 and patched in the same month. CVE-2023-29324 was due to the way in which *Windows* handles UNC paths and path conversions, but an in-depth explanation of this vulnerability is beyond the scope of this paper.

3.2 CVE-2023-21716

CVE-2023-21716 is a heap corruption vulnerability reported in *MS Word*'s RTF (Rich Text Format) file parser, leading to a remote code execution (RCE). This vulnerability was assigned a CVSS score of 9.8.

How can a *Word* application vulnerability impact *Outlook*? Well, *Outlook* has a preview pane which conveniently allows one to preview the documents attached to the email. *Outlook* uses the same library to preview RTF files as is used by *MS Word* to parse these files, allowing *Outlook* to become a target as well.

Exploitation of the RTF parser is not new. CVE-2010-3333 [12], CVE-2014-1761 [13] and CVE-2017-0199 are a few of the vulnerabilities in the RTF parser which have been exploited in the past. Some of the vulnerabilities are due to how the parser handles the RTF file format, whilst others are related to how OLE-type objects are handled in RTF.

`Wwlib.dll` is responsible for parsing RTF files, and is imported by *MS Word*, *Explorer* and *Outlook* (for document preview pane viewing), so all these applications are impacted by the issue in the DLL. A threat actor only needs to trick the user into previewing an attached crafted RTF file to trigger the exploitation.

A Note on the RTF file format

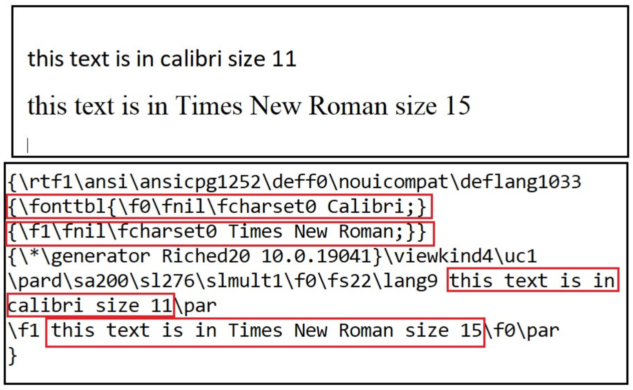
RTF is a proprietary specification released by *MS* back in 1987 and updated until 2008. RTF was released to support document exchange between various word processing applications on different platforms. The last specification document, MSFT-RTF (RTF version 1.9.1), was released in 2008 [14].

An RTF file consists of Control Information, which includes Control Words and Groups enclosed in curly braces. Control Words are words or commands used to define properties of sections in a document. These begin with backslash ('\') and are of the following format:

```
\<ASCII Character Sequence><Delimited>
```

Some Control Words end with a numeric digit or a minus ('-') sign, indicating a numeric parameter. This parameter can be a 16-bit signed integer (i.e. -32768 to 32767).

Groups are sets of Control Words, text and Control Symbols enclosed within curly braces ('`{ }`'). Text within each Group is impacted by the properties set by Control Words and Control Symbols in that Group.


this text is in calibri size 11

this text is in Times New Roman size 15

```
{\rtf1\ansi\ansicpg1252\deff0\nouicompat\deflang1033
{\fonttbl{\f0\fnil\fcharset0 Calibri;}
{\f1\fnil\fcharset0 Times New Roman;}}
{\*\generator Riched20 10.0.19041}\viewkind4\uc1
\pard\sa200\sl276\slmult1\f0\fs22\lang9 this text is in
calibri size 11\par
\f1 this text is in Times New Roman size 15\f0\par
}
```

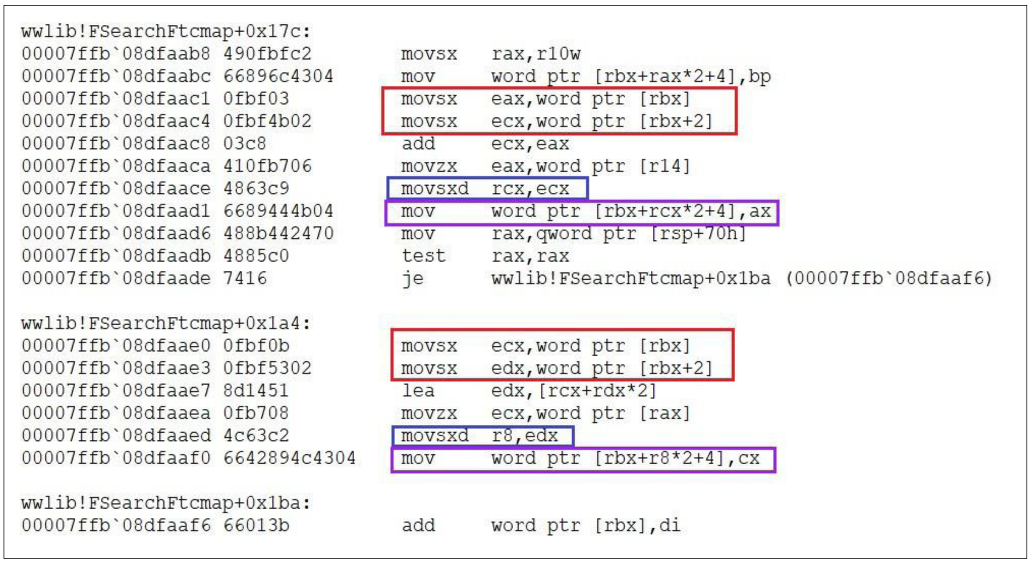
Figure 15: Visible content of RTF file and representation in RTF file structure.

Figure 15 shows the content of an RTF file generated using *WordPad* in *Windows 10* and its associated file structure. We can see Control Words such as `\rtf1`, `\fonttbl`, `\par`, `\f0`, etc. used in the file in different places.

The `\fonttbl` Control Word represents the font table that lists the fonts that can be referred to in the document. In most of the cases, only the fonts that are actually used in the document are listed here. Each font follows the Control Word `\f<num>`, where `<num>` indicates the Font ID. This control word is then used to refer to the font in the relevant sections that follow. In the above example, the font table has two fonts: `\f0`, Calibiri, and `\f1`, Times New Roman. These are then referred to before the text using the Control Words.

Immediately after CVE-2023-21716 was patched in February 2023, a proof of concept for the vulnerability was published [15]. An associated Python script shared online generated a malicious RTF document which crashed the application when accessed in *MS Word*. Upon analysing the script and the generated malicious document, it was observed that when a long list of fonts is included in the font table, it is possible to crash winword.exe as soon as the document is accessed. In the case of *Outlook*, as soon as the document was previewed, *Outlook* crashed. We all know that such manner of crashing could provide an opportunity for exploitation.

So, what's the mechanism for the flaw? While parsing the RTF file, a call is made to the `wwlib!FSearchFtcmap` API. This function is where the flaw exists. Setting a breakpoint in the function call when font ID equals 0x7FF7, we observe the code in Figure 16.



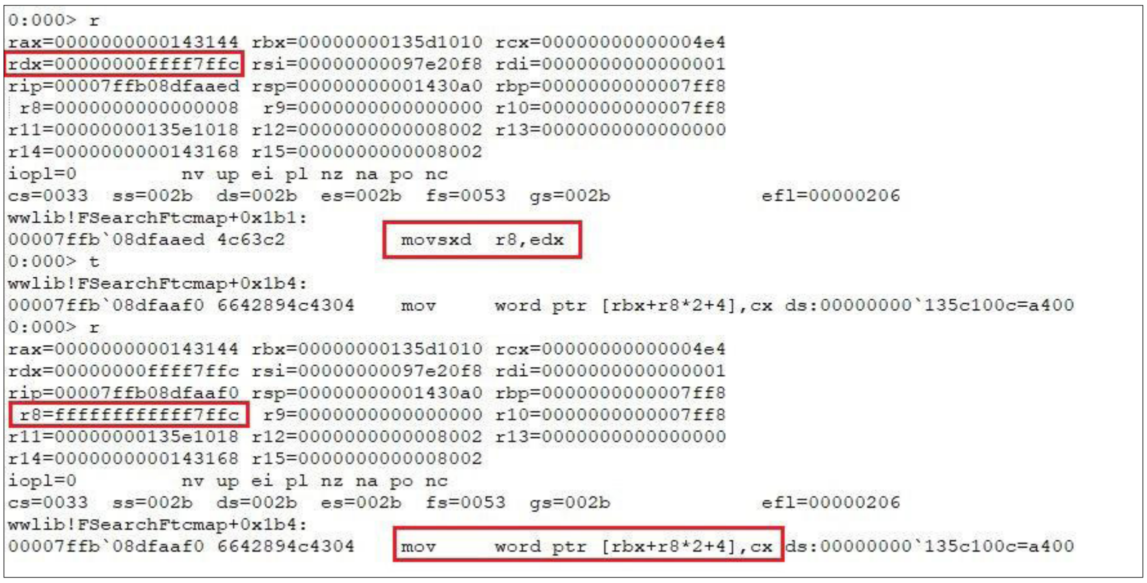
```
wwlib!FSearchFtcmap+0x17c:
00007ffb`08dfaab8 490fbfc2        movsx   rax,r10w
00007ffb`08dfaabc 66896c4304      mov     word ptr [rbx+rax*2+4],bp
00007ffb`08dfaac1 0fbf03          movsx   eax,word ptr [rbx]
00007ffb`08dfaac4 0fbf4b02        movsx   ecx,word ptr [rbx+2]
00007ffb`08dfaac8 03c8            add     ecx,eax
00007ffb`08dfaaca 410fb706        movzx   eax,word ptr [r14]
00007ffb`08dfaace 4863c9          movsxd  rcx,ecx
00007ffb`08dfaad1 6689444b04      mov     word ptr [rbx+rcx*2+4],ax
00007ffb`08dfaad6 488b442470      mov     rax,qword ptr [rsp+70h]
00007ffb`08dfaadb 4885c0          test    rax,rax
00007ffb`08dfaade 7416            je      wwlib!FSearchFtcmap+0x1ba (00007ffb`08dfaaf6)

wwlib!FSearchFtcmap+0x1a4:
00007ffb`08dfaae0 0fbf0b          movsx   ecx,word ptr [rbx]
00007ffb`08dfaae3 0fbf5302        movsx   edx,word ptr [rbx+2]
00007ffb`08dfaae7 8d1451          lea     edx,[rcx+rdx*2]
00007ffb`08dfaaea 0fb708          movzx   ecx,word ptr [rax]
00007ffb`08dfaaed 4c63c2          movsxd  r8,edx
00007ffb`08dfaaf0 6642894c4304    mov     word ptr [rbx+r8*2+4],cx

wwlib!FSearchFtcmap+0x1ba:
00007ffb`08dfaaf6 66013b          add     word ptr [rbx],di
```

Figure 16: Disassembled code from wwlib!FSearchFtcmap.

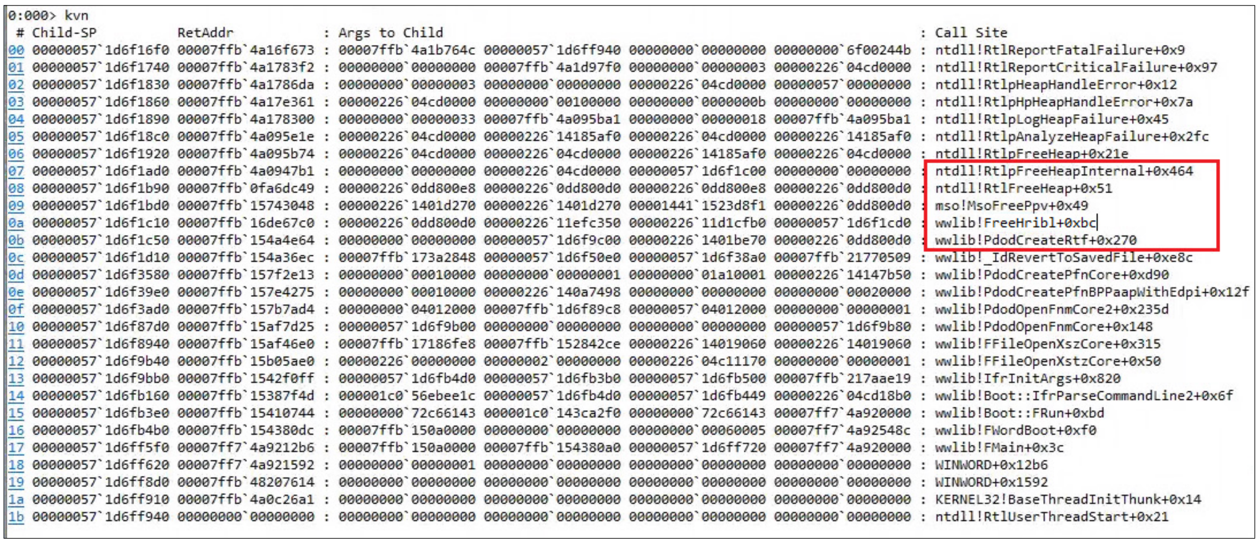
As seen in Figure 16, values moved to registers are sign extended using the `movsx` and `movsxd` instructions (highlighted in red and blue, respectively).



```
0:000> r
rax=0000000000143144 rbx=00000000135d1010 rcx=00000000000004e4
rdx=00000000ffff7ffc rsi=00000000097e20f8 rdi=0000000000000001
rip=00007ffb08dfaaed rsp=00000000001430a0 rbp=0000000000007ff8
 r8=0000000000000008  r9=0000000000000000 r10=0000000000007ff8
r11=00000000135e1018 r12=0000000000008002 r13=0000000000000000
r14=0000000000143168 r15=0000000000008002
iopl=0         nv up ei pl nz na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
wwlib!FSearchFtcmap+0x1b1:
00007ffb`08dfaaed 4c63c2          movsxd  r8,edx
0:000> t
wwlib!FSearchFtcmap+0x1b4:
00007ffb`08dfaaf0 6642894c4304    mov     word ptr [rbx+r8*2+4],cx ds:00000000`135c100c=a400
0:000> r
rax=0000000000143144 rbx=00000000135d1010 rcx=00000000000004e4
rdx=00000000ffff7ffc rsi=00000000097e20f8 rdi=0000000000000001
rip=00007ffb08dfaaf0 rsp=00000000001430a0 rbp=0000000000007ff8
 r8=ffffffffffff7ffc  r9=0000000000000000 r10=0000000000007ff8
r11=00000000135e1018 r12=0000000000008002 r13=0000000000000000
r14=0000000000143168 r15=0000000000008002
iopl=0         nv up ei pl nz na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
wwlib!FSearchFtcmap+0x1b4:
00007ffb`08dfaaf0 6642894c4304    mov     word ptr [rbx+r8*2+4],cx ds:00000000`135c100c=a400
```

Figure 17: Snippet from debugging.

Figure 17 shows the output of the `movsxd` instruction. As a result of this, the calculations (highlighted in purple in Figure 16) would result in an overflow, writing data in incorrect memory locations, resulting in heap corruption. The program continues its execution even after the write, and crashes subsequently in *`wwlib!FreeHribl`* due to this heap corruption (Figure 18).



```
0:000> kvn
 # Child-SP          RetAddr           : Args to Child                                                           : Call Site
00 00000057`1d6f16f0 00007ffb`4a16f673 : 00007ffb`4a1b764c 00000057`1d6ff940 00000000`00000000 00000000`6f00244b : ntdll!RtlReportFatalFailure+0x9
01 00000057`1d6f1740 00007ffb`4a1783f2 : 00000000`00000000 00007ffb`4a1d97f0 00000000`00000003 00000226`04cd0000 : ntdll!RtlReportCriticalFailure+0x97
02 00000057`1d6f1830 00007ffb`4a1786da : 00000000`00000003 00000000`00000000 00000226`04cd0000 00000057`00000000 : ntdll!RtlpHeapHandleError+0x12
03 00000057`1d6f1860 00007ffb`4a17e361 : 00000226`04cd0000 00000000`00100000 00000000`0000000b 00000000`00000000 : ntdll!RtlpHpHeapHandleError+0x7a
04 00000057`1d6f1890 00007ffb`4a178300 : 00000000`00000033 00007ffb`4a095ba1 00000000`00000018 00007ffb`4a095ba1 : ntdll!RtlpLogHeapFailure+0x45
05 00000057`1d6f18c0 00007ffb`4a095e1e : 00000226`04cd0000 00000226`14185af0 00000226`04cd0000 00000226`14185af0 : ntdll!RtlpAnalyzeHeapFailure+0x2fc
06 00000057`1d6f1920 00007ffb`4a095b74 : 00000226`04cd0000 00000226`04cd0000 00000226`14185af0 00000226`04cd0000 : ntdll!RtlpFreeHeap+0x21e
07 00000057`1d6f1ad0 00007ffb`4a0947b1 : 00000000`00000000 00000226`04cd0000 00000057`1d6f1c00 00000000`00000000 : ntdll!RtlpFreeHeapInternal+0x464
08 00000057`1d6f1b90 00007ffb`0fa6dc49 : 00000226`0dd800e8 00000226`0dd800d0 00000226`0dd800e8 00000226`0dd800d0 : ntdll!RtlFreeHeap+0x51
09 00000057`1d6f1bd0 00007ffb`15743048 : 00000226`1401d270 00000226`1401d270 00001441`1523d8f1 00000226`0dd800d0 : mso!MsoFreePpv+0x49
0a 00000057`1d6f1c10 00007ffb`16de67c0 : 00000226`0dd800d0 00000226`11efc350 00000226`11d1cfb0 00000057`1d6f1cd0 : wwlib!FreeHribl+0xbc
0b 00000057`1d6f1c50 00007ffb`154a4e64 : 00000000`00000000 00000057`1d6f9c00 00000226`1401be70 00000226`0dd800d0 : wwlib!PdodCreateRtf+0x270
0c 00000057`1d6f1d10 00007ffb`154a36ec : 00007ffb`173a2848 00000057`1d6f50e0 00000057`1d6f38a0 00007ffb`21770509 : wwlib!_IdRevertToSavedFile+0xe8c
0d 00000057`1d6f3580 00007ffb`157f2e13 : 00000000`00010000 00000000`00000001 00000000`01a10001 00000226`14147b50 : wwlib!PdodCreatePfnCore+0xd90
0e 00000057`1d6f39e0 00007ffb`157e4275 : 00000000`00010000 00000226`140a7498 00000000`00000000 00000000`00020000 : wwlib!PdodCreatePfnBPPaapWithEdpi+0x12f
0f 00000057`1d6f3ad0 00007ffb`157b7ad4 : 00000000`04012000 00007ffb`1d6f89c8 00000057`04012000 00000000`00000001 : wwlib!PdodOpenFnmCore2+0x235d
10 00000057`1d6f87d0 00007ffb`15af7d25 : 00000057`1d6f9b00 00000000`00000000 00000000`00000000 00000057`1d6f9b80 : wwlib!PdodOpenFnmCore+0x148
11 00000057`1d6f8940 00007ffb`15af46e0 : 00007ffb`17186fe8 00007ffb`152842ce 00000226`14019060 00000226`14019060 : wwlib!FFileOpenXszCore+0x315
12 00000057`1d6f9b40 00007ffb`15b05ae0 : 00000226`00000000 00000002`00000000 00000226`04c11170 00000000`00000001 : wwlib!FFileOpenXstzCore+0x50
13 00000057`1d6f9bb0 00007ffb`1542f0ff : 00000057`1d6fb4d0 00000057`1d6fb3b0 00000057`1d6fb500 00007ffb`217aae19 : wwlib!IfrInitArgs+0x820
14 00000057`1d6fb160 00007ffb`15387f4d : 000001c0`56ebee1c 00000057`1d6fb4d0 00000057`1d6fb449 00000226`04cd18b0 : wwlib!Boot::IfrParseCommandLine2+0x6f
15 00000057`1d6fb3e0 00007ffb`15410744 : 00000000`72c66143 000001c0`143ca2f0 00000000`72c66143 00007ff7`4a920000 : wwlib!Boot::FRun+0xbd
16 00000057`1d6fb4b0 00007ffb`154380dc : 00007ffb`150a0000 00000000`00000000 00000000`00060005 00007ff7`4a92548c : wwlib!FWordBoot+0xf0
17 00000057`1d6ff5f0 00007ff7`4a9212b6 : 00007ffb`150a0000 00007ffb`154380a0 00000057`1d6ff720 00007ff7`4a920000 : wwlib!FMain+0x3c
18 00000057`1d6ff620 00007ff7`4a921592 : 00000000`00000001 00000000`00000000 00000000`00000000 00000000`00000000 : WINWORD+0x12b6
19 00000057`1d6ff8d0 00007ffb`48207614 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : WINWORD+0x1592
1a 00000057`1d6ff910 00007ffb`4a0c26a1 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : KERNEL32!BaseThreadInitThunk+0x14
1b 00000057`1d6ff940 00000000`00000000 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!RtlUserThreadStart+0x21
```

Figure 18: Stack trace at the time of crash.

To patch the vulnerability, *MS* added a function call to `SafeIntOnOverflow` from the SafeInt class, in the `wwlib!FSearchFtcmap` API, as shown in Figure 19.

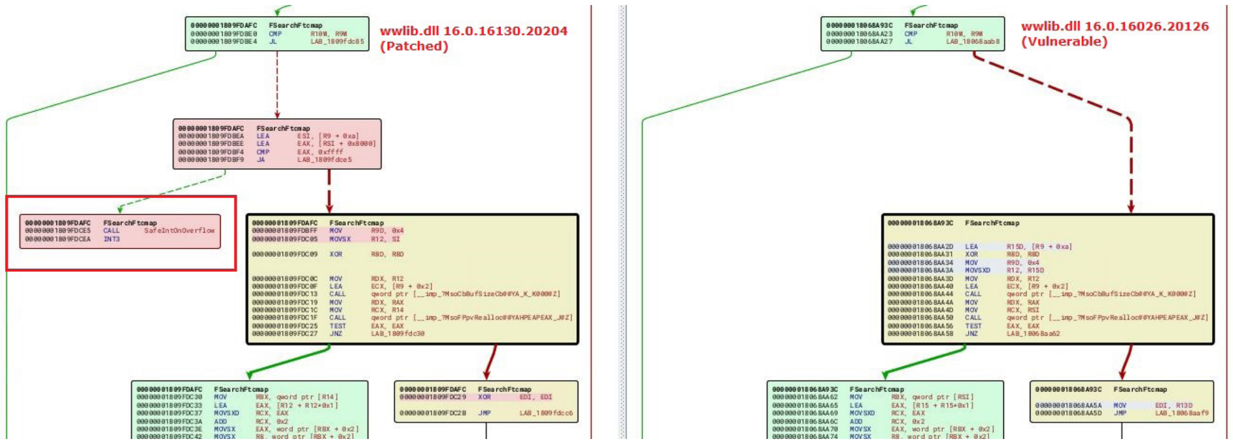
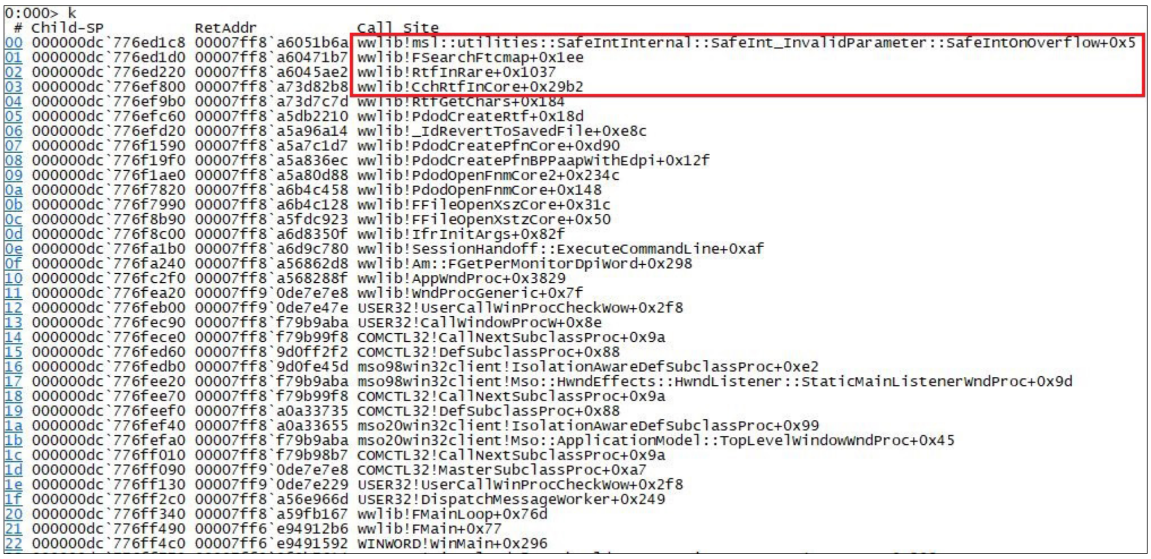

wwlib.dll 16.0.16130.20204 (Patched)
CALL SafeIntOnOverflow
wwlib.dll 16.0.16026.20126 (Vulnerable)

Figure 19: Patch diffing showing the added function call (left).

It compares the value and calls the API to handle the overflow gracefully by safely catching the overflow, as shown in Figure 20.



```
0:000> k
 # Child-SP          RetAddr           Call Site
00 000000dc`776ed1c8 00007ff8`a6051b6a wwlib!msl::utilities::SafeIntInternal::SafeInt_InvalidParameter::SafeIntOnOverflow+0x5
01 000000dc`776ed1d0 00007ff8`a60471b7 wwlib!FSearchFtcmap+0x1ee
02 000000dc`776ed220 00007ff8`a6045ae2 wwlib!RtfInRare+0x1037
03 000000dc`776ef800 00007ff8`a73d82b8 wwlib!CchRtfInCore+0x29b2
04 000000dc`776ef9b0 00007ff8`a73d7c7d wwlib!RtfGetChars+0x184
05 000000dc`776efc60 00007ff8`a5db2210 wwlib!PdodCreateRtf+0x18d
06 000000dc`776efd20 00007ff8`a5a96a14 wwlib!_IdRevertToSavedFile+0xe8c
07 000000dc`776f1590 00007ff8`a5a7c1d7 wwlib!PdodCreatePfnCore+0xd90
08 000000dc`776f19f0 00007ff8`a5a836ec wwlib!PdodCreatePfnBPPaapWithEdpi+0x12f
09 000000dc`776f1ae0 00007ff8`a5a80d88 wwlib!PdodOpenFnmCore2+0x234c
0a 000000dc`776f7820 00007ff8`a6b4c458 wwlib!PdodOpenFnmCore+0x148
0b 000000dc`776f7990 00007ff8`a6b4c128 wwlib!FFileOpenXszCore+0x31c
0c 000000dc`776f8b90 00007ff8`a5fdc923 wwlib!FFileOpenXstzCore+0x50
0d 000000dc`776f8c00 00007ff8`a6d8350f wwlib!IfrInitArgs+0x82f
0e 000000dc`776fa1b0 00007ff8`a6d9c780 wwlib!SessionHandoff::ExecuteCommandLine+0xaf
0f 000000dc`776fa240 00007ff8`a56862d8 wwlib!Am::FGetPerMonitorDpiWord+0x298
10 000000dc`776fc2f0 00007ff8`a568288f wwlib!AppWndProc+0x3829
11 000000dc`776fea20 00007ff9`0de7e7e8 wwlib!WndProcGeneric+0x7f
12 000000dc`776feb00 00007ff9`0de7e47e USER32!UserCallWinProcCheckWow+0x2f8
13 000000dc`776fec90 00007ff8`f79b9aba USER32!CallWindowProcW+0x8e
14 000000dc`776fece0 00007ff8`f79b99f8 COMCTL32!CallNextSubclassProc+0x9a
15 000000dc`776fed60 00007ff8`9d0ff2f2 COMCTL32!DefSubclassProc+0x88
16 000000dc`776fedb0 00007ff8`9d0fe45d mso98win32client!IsolationAwareDefSubclassProc+0xe2
17 000000dc`776fee20 00007ff8`f79b9aba mso98win32client!Mso::HwndEffects::HwndListener::StaticMainListenerWndProc+0x9d
18 000000dc`776fee70 00007ff8`f79b99f8 COMCTL32!CallNextSubclassProc+0x9a
19 000000dc`776feef0 00007ff8`a0a33735 COMCTL32!DefSubclassProc+0x88
1a 000000dc`776fef40 00007ff8`a0a33655 mso20win32client!IsolationAwareDefSubclassProc+0x99
1b 000000dc`776fefa0 00007ff8`f79b9aba mso20win32client!Mso::ApplicationModel::TopLevelWindowWndProc+0x45
1c 000000dc`776ff010 00007ff8`f79b98b7 COMCTL32!CallNextSubclassProc+0x9a
1d 000000dc`776ff090 00007ff9`0de7e7e8 COMCTL32!MasterSubclassProc+0xa7
1e 000000dc`776ff130 00007ff9`0de7e229 USER32!UserCallWinProcCheckWow+0x2f8
1f 000000dc`776ff2c0 00007ff8`a56e966d USER32!DispatchMessageWorker+0x249
20 000000dc`776ff340 00007ff8`a59fb167 wwlib!FMainLoop+0x76d
21 000000dc`776ff490 00007ff6`e94912b6 wwlib!FMain+0x77
22 000000dc`776ff4c0 00007ff6`e9491592 WINWORD!WinMain+0x296
```

Figure 20: Stack trace when application crashes in the patched version.

4. OUTLOOK FOR THE FUTURE

In January 2021, ProxyLogon (CVE-2021-26855) [16] was reported in *MS Exchange Server*, which was a server-side request forgery vulnerability leading to authenticated bypass. This was chained with other vulnerabilities reported at the same time to gain access to the exchange server and the network. ProxyLogon and its associated vulnerabilities were heavily exploited in the wild. This opened the floodgates for vulnerabilities reported in *Exchange Server*. Up until 2022 at least eight vulnerabilities were reported in *Exchange* servers and some of them were exploited in the wild. In addition, NTLM relay attacks are not new to *Windows*. Vulnerabilities disclosed in *Windows* such as CVE-2021-36942 (PetitPotam) [17] and the presence of a myriad of write-ups on NTLM relay and tools are testament to this. One of the vulnerabilities reported in *Exchange*, CVE-2021-33768, was an NTLM relay to the *Exchange Server* front-end vulnerability. So, can we predict what’s to come for *Outlook*?

CVE-2023-23397, which can be exploited remotely without authentication and user interaction, to leak NTLM tokens, can be used to gain an initial foothold in the network. At the very least, it can be used to access and modify user mailbox permissions, which is what was observed in the wild by *Microsoft*. This is certainly something to ‘look out’ for.

Earlier, macros in *Office* documents allowed adversaries to easily gain persistent access to the target system; but with those almost out of commission, adversaries will now look to weaponize vulnerabilities in *Office* applications for the same end. CVE-2023-21716 could allow them to do exactly that, with the added advantage that exploitation can be achieved merely by previewing the exploit document.

Consequently, we should be prepared for weaponized documents in spear-phishing mails, this time exploiting *Outlook* vulnerabilities, including even meeting invites.

Mitigations

Along with the patches for the discussed vulnerabilities, *MS* released advisories to protect against possible similar attacks. Some suggestions are outlined below:

- CVE-2023-23397
    - Block SMB outbound traffic to untrusted networks.
    - Mitigate pass-the-hash attacks [18].
    - Use Kerberos wherever possible instead of NTLM.
- CVE-2023-21716
    - Block RTF documents from unknown or untrusted sources.

In addition to blocking exploits and attempted exploitation methods, security product vendors should also make customers aware of such vulnerabilities and flag vulnerable components present in an organization’s systems and networks.

REFERENCES

[1] Microsoft Windows Support Diagnostic Tool (MSDT) Remote Code Execution Vulnerability CVE-2022-30190. Microsoft. https://msrc.microsoft.com/update-guide/vulnerability/CVE-2022-30190.

[2] Shandilya, A. The Secrets of Trident. K7 Security Labs. 28 February 2022. https://labs.k7computing.com/index.php/the-secrets-of-trident/.

[3] Microsoft Outlook Elevation of Privilege Vulnerability CVE-2023-23397. Microsoft. https://msrc.microsoft.com/update-guide/vulnerability/CVE-2023-23397.

[4] Microsoft Word Remote Code Execution Vulnerability CVE-2023-21716. Microsoft. https://msrc.microsoft.com/update-guide/vulnerability/CVE-2023-21716.

[5] MAPI architecture overview. Microsoft. 3 March 2022. https://learn.microsoft.com/en-us/office/client-developer/outlook/mapi/mapi-architecture-overview.

[6] stephenegriffin. mfcmapi. https://github.com/stephenegriffin/mfcmapi.

[7] OutlookSpy 5.0. https://www.dimastr.com/outspy/home.htm.

[8] Microsoft Incident Response. Guidance for investigating attacks using CVE-2023-23397. Microsoft. 24 March 2023. https://www.microsoft.com/en-us/security/blog/2023/03/24/guidance-for-investigating-attacks-using-cve-2023-23397.

[9] Microsoft. [MS-OXORMDR]: Reminder Settings Protocol. 17 August 2021. https://learn.microsoft.com/en-us/openspecs/exchange_server_protocols/ms-oxormdr/5454ebcc-e5d1-4da8-a598-d393b101caab.

[10] Microsoft. CSS-Exchange. CVE-2023-23397 script. https://microsoft.github.io/CSS-Exchange/Security/CVE-2023-23397/.

[11] Barnea, B. From One Vulnerability to Another: Outlook Patch Analysis Reveals Important Flaw in Windows API. Akamai. 10 May 2023. https://www.akamai.com/blog/security-research/important-outlook-vulnerability-bypass-windows-api.

[12] Yang, J. How RTF malware evades static signature-based detection. Mandiant. 20 May 2016. https://www.mandiant.com/resources/blog/how-rtf-malware-evad.

[13] Li, H. RTF Zero-Day Attack CVE-2014-1761 Shows Sophistication of Attackers. McAfee. 3 April 2014. https://www.mcafee.com/blogs/other-blogs/mcafee-labs/close-look-rtf-zero-day-attack-cve-2014-1761-shows-sophistication-attackers/.

[14] Microsoft Corporation. Rich Text Format (RTF) Specification Version 1.9.1. https://interoperability.blob.core.windows.net/files/Archive_References/[MSFT-RTF].pdf.

[15] jduck. Microsoft Word RTF Font Table Heap Corruption. 20 November 2022. https://qoop.org/publications/cve-2023-21716-rtf-fonttbl.md.

[16] Tsai, O. A New Attack Surface on MS Exchange Part 1 - ProxyLogon! DEVCORE. 19 October 2022. https://devco.re/blog/2021/08/06/a-new-attack-surface-on-MS-exchange-part-1-ProxyLogon/.

[17] Shandilya, A. The PetitPotam Story. K7 Security Labs. 1 October 2021. https://labs.k7computing.com/index.php/the-petitpotam-story/.

[18] Trustworthy Computing. Mitigating Pass-the-Hash and Other Credential Theft, version 2. Microsoft. https://download.microsoft.com/download/7/7/a/77abc5bd-8320-41af-863c-6ecfb10cb4b9/mitigating-pass-the-hash-attacks-and-other-credential-theft-version-2.pdf.

[19] Shandilya, A. CVE-2023-21716: A new Office Exploit. K7 Security Labs. 22 March 2023. https://labs.k7computing.com/index.php/cve-2023-21716-a-new-office-exploit/.

[20] MDSec. Exploiting CVE-2023-23397: Microsoft Outlook Elevation of Privilege Vulnerability. 14 March 2023. https://www.mdsec.co.uk/2023/03/exploiting-cve-2023-23397-microsoft-outlook-elevation-of-privilege-vulnerability/.

[21] Unit 42. Threat Brief - CVE-2023-23397 - Microsoft Outlook Privilege Escalation. 31 March 2023. https://unit42.paloaltonetworks.com/threat-brief-cve-2023-23397/.

[22] Penetration Testing Lab. Microsoft Exchange – NTLM Relay. 9 September 2019. https://pentestlab.blog/2019/09/09/microsoft-exchange-ntlm-relay/.

COMPONENT AND SAMPLE HASHES

S no	File name	Hash (SHA256)
1	Outlook.EXE (16.0.16026.21146)	d642c1d4e97477263d17b705467816d4ed625ea275607297735cf0743f4a07aa
2	Outlook.EXE (16.0.1602630.20332)	45a66726915893e4b0bd56aba177c244b13da3344949377140a29df8e7c9ba13
3	WWLIB.DLL (16.0.16130.20204)	030133f9a264ca7fa46e4f2f91c88f765f3b9ffd9d901bdf52beaad26ac71e60
4	WWLIB.DLL (16.0.16026.20126)	3808a7f7c40d89de8d6e26156c3157fe94d809966c8b05cd5442ba3c3e625300
5	test.msg	7d94c1946abe60549dd724309257095f96d4a41784e22e7fdd2821048e666151