

New Updates to ValleyRAT | ThreatLabz

By Muhammed Irfan V A, Manisha Ramcharan Prajapati

Published: 2024-06-10 · Archived: 2026-04-05 14:07:49 UTC

Technical Analysis

The campaign we analyzed delivers ValleyRAT as the payload in the final stage. The figure below illustrates the attack chain for this particular campaign.

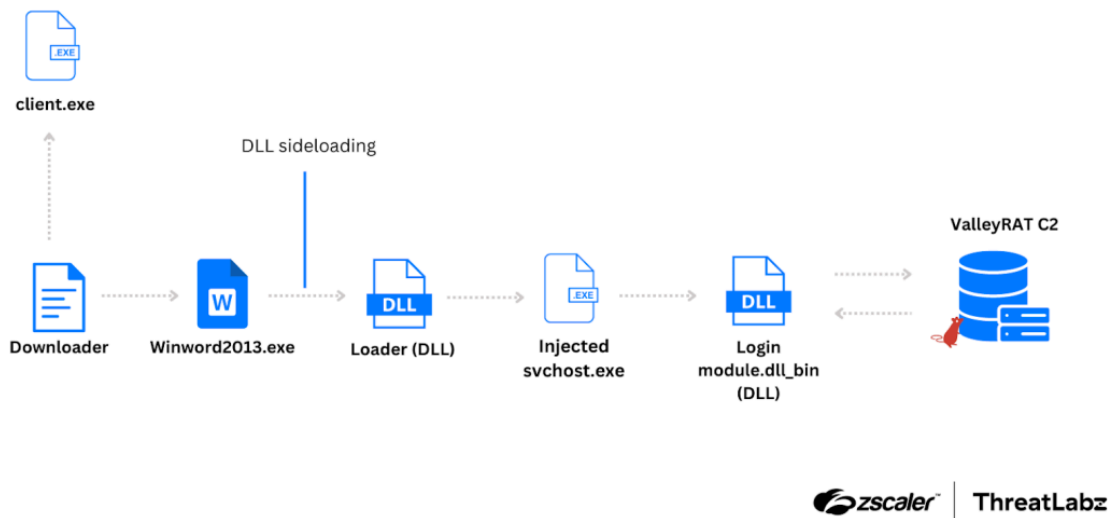
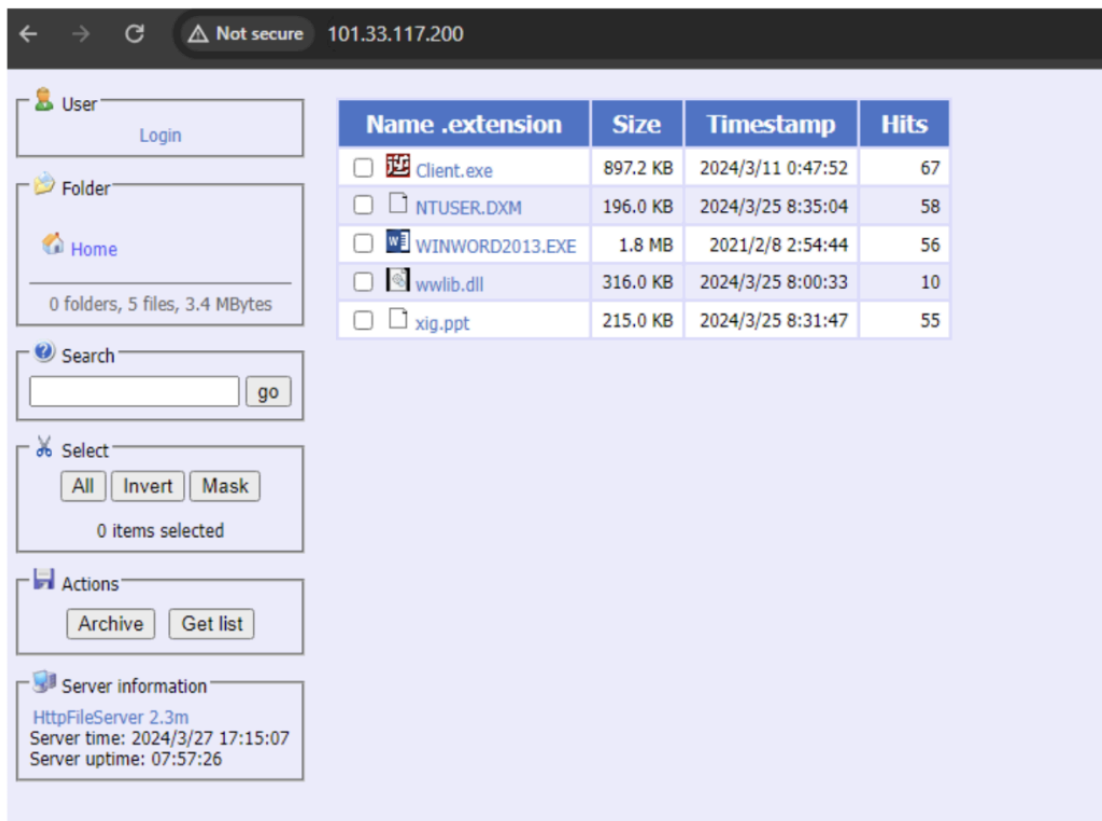


Figure 1: Attack chain for the campaign, where ValleyRAT is delivered as the payload in the final stage.

First stage

Downloader

ValleyRAT uses an initial stage downloader that proceeds to retrieve five files from an HFS server (that is also used later for C2 communications), as shown in the figure below.




 ThreatLabz

Figure 2: HFS server hosting second stage files for ValleyRAT.

The downloader first checks for the presence of the file `NTUSER.DXM`. If the file is not found, the malware downloads it from the web and saves it to disk using the following APIs:

- `URLOpenBlockingStreamW` - Utilized to download the files as an `IStream`.
- `SHCreateStreamOnFileEx` - Used to create a file and write the downloaded `IStream` into it.

The downloaded file, `NTUSER.DXM`, is then decrypted using a combination of XOR decryption and RC4 decryption. The XOR key `[9F 4B 27 D3 51 8E CD 2A BF 3C A1 56 E4 78 9A 3D]` and RC4 key `[21 72 53 14 85 96 A7 B8 C9 DA EB FC 0D 1E 2F 30]` are loaded as stack strings.

The code sample below shows the decryption algorithm replicated in Python.

```
from Crypto.Cipher import ARC4

def xor_decrypt(ciphertext, xor_key, key_length):
    decrypted = bytearray()
    for i, byte in enumerate(ciphertext):
        decrypted.append(byte ^ xor_key[i % key_length])
    return bytes(decrypted)

def rc4_decrypt(ciphertext, rc4_key):
    cipher = ARC4.new(rc4_key)
    decrypted = cipher.decrypt(ciphertext)
    return decrypted

def decrypt_file(filename, xor_key, xor_key_length, rc4_key):
    with open(filename, 'rb') as file:
        ciphertext = file.read()
```

```
xor_decrypted = xor_decrypt(ciphertext, xor_key, xor_key_length)
decrypted_payload = rc4_decrypt(xor_decrypted, rc4_key)
with open("second_stage_sample.bin", 'ab') as write_file:
    write_file.write(decrypted_payload)
print("[+] Second stage successfully written to disk as second_stage_sample.bin")
```

The file decrypted using the algorithm above is a DLL. Once the DLL is decrypted, the malware invokes the export function `_MainLogic@0` from within the DLL file.

The decrypted DLL first checks for the existence of the path `C:\Program Files\TCLS`. If the path does not exist, it proceeds to download `client.exe` from the HFS server using the WinINet library, with `Processkiller` set as the `UserAgent`.

Anti-AV checks

The decrypted DLL includes an anti-AV check to detect, and terminate Qihoo security software and the `Winrar` utility. It retrieves a list of all processes running on the system and compares the process names with the names below:

- `ZhuDongFangYu`
- `SoftMgrLite`
- `DumpUper`
- `Winrar`
- `safesvr`

The process names `ZhuDongFangYu`, `SoftMgrLite`, `DumpUper`, and `safesvr` are associated with Qihoo security software. We suspect that ValleyRAT is terminating `Winrar` due to its ability to integrate with antivirus software to scan archive files for malicious content. Previous campaigns have utilized zipped executables as first stage downloaders, which may explain this behavior. If a process name matches, the malware opens a handle to the process and sends a `WM_QUIT` message to all the threads within the process, effectively terminating them.

Following this, the malware downloads `WINWORD2013.EXE`, `wwlib.dll`, and `xig.ppt` from the HFS server, saving them to the disk at the location `C:\Users`.

The malware deletes the directory `C:\Program Files\TCLS` and the file `client.exe`.

Finally, the malware attempts to execute `WINWORD2013.EXE` with administrative privileges using the `runas` command, leading to the second stage.

Second stage

Loader (wwlib.dll)

The file `WINWORD2013.EXE` is the legitimate Microsoft Word processor. However, the malware utilizes it to sideload a malicious DLL called `wwlib.dll`. The `wwlib.dll` serves as a malicious loader, responsible for checking the presence of `C:\Users\xig.ppt` (an encrypted DLL) on the disk. If the file is found, the malware loads it into memory and decrypts it using the same decryption algorithm mentioned in the first stage using the same XOR and RC4 keys. The malware copies the decrypted `xig.ppt` DLL to another memory location with `PAGE_EXECUTE_READ` permission.

Process injection

From here, the decrypted `xig.ppt` continues the execution process as a mechanism to decrypt and inject shellcode into `svchost.exe`.

The malware creates `svchost.exe` as a suspended process, allocates memory within the process, and writes shellcode there. The malware uses the `SetThreadContext` API to change the instruction pointer to the address of the allocated shellcode.

Finally, the malware calls the `ResumeThread` function, leading to the next stage of the process. The figure below shows the decompiled code the malware uses for injection.

```
BVar4 = GetThreadContext(puVar12, (LPCONTEXT)ThreadContext);
if ((BVar4 != 0) &&
    (lpBaseAddress = VirtualAllocEx(processHandle, (LPVOID)0x0, 0x98b, 0x3000, PAGE_EXECUTE_READWRITE),
    lpBaseAddress != (LPVOID)0x0)) {
    BVar4 = WriteProcessMemory(processHandle, lpBaseAddress, &lpBuffer, 0x98b, (SIZE_T *)&local_24);
    if ((BVar4 != 0) &&
        (local_280 = lpBaseAddress, BVar4 = SetThreadContext(local_1394, (CONTEXT *)&ThreadContext),
        BVar4 != 0)) {
        ResumeThread(processInformation.hThread);
    }
}
```

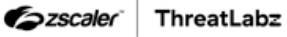


Figure 3: Process injection used in the second stage.

Persistence

The second stage is also responsible for establishing persistence. The malware accomplishes this by adding `C:\Users\WINWORD2013.EXE` to the autorun key `Software\Microsoft\Windows\CurrentVersion\Run` with the name “`WINWORD2013`”.

Additionally, the malware sets the attributes of `WINWORD2013.EXE`, `wwlib.dll`, and `xig.ppt` to `FILE_ATTRIBUTE_SYSTEM` | `FILE_ATTRIBUTE_HIDDEN`.

Third stage

Injected shellcode

The shellcode injected contains essential configuration information and resolves APIs to establish a connection with the C2 server. This connection is utilized to download the next stage of the malware.

Dynamic API resolving

The shellcode injected into `svchost.exe` dynamically resolves APIs by traversing the Process Environment Block (PEB) and parsing PE headers using the BKDR hashing algorithm below.

```
def BKDRHashing(apiName):
    finalHash = 0
    for i in apiName:
        finalHash = (finalHash* 0x83) & 0xFFFFFFFF
        finalHash = (finalHash + ord(i)) & 0xFFFFFFFF
    finalHash = finalHash & 0x7FFFFFFF
    print(hex(finalHash))

>>>BKDRHashing("GetProcAddress")
0x1ab9b854
```

Configuration format

Offset	Description	Example Value
0x28	Boolean value. If the value is 0, ValleyRAT utilizes UDP for C2 communication [Option 1]. If the value is 1, it employs TCP for C2 communication [Option 1].	0x1
0x2c	C2 IP length [Option 2].	0xF
0x30	C2 port [Option 2] stored as 16-bit number in host byte order.	0x22B8 (8888)
0x34	Boolean value. If the value is 0, ValleyRAT utilizes UDP for C2 communication [Option 2]. If the value is 1, it employs TCP for C2 communication [Option 2].	0x1
0x38	C2 IP data buffer [Option 1].	101.33.117.200

Offset	Description	Example Value
0x38 + Value Stored in offset 0x20	C2 IP data buffer [Option 2].	101.33.117.200
0x38 + ((value stored in offset 0x20) * 2)	The configuration string is stored in reverse , where p1 , o1 , p2 , and o2 are related to C2 communication (explained in the next section). The values of c1 and dd are multiplied by 1000 and used as arguments for the sleep function.	0:db 0:lk 0:hs 0:ld 0:ll 0:hb 0:pj 02.3.4202:zb 0.1:bb 认 默:zf 1:lc 1:dd 1:3t 08:3o 1.0.0.721:3p 1:2t 8888:2o 002.711.33.101:2p 1:1t 6666:1o 002.711.33.10

Table 1: The configuration format used for ValleyRAT C2 communication.

The sample analyzed utilizes TCP for communication with the C2 server. Subsequently, the malware sends the data 32 to the C2 in order to receive a 32-bit shellcode. We confirmed this by sending data as 64 and receiving a 64-bit shellcode. The 32-bit shellcode is received as encrypted data with a size of 0x4B00E. The encrypted data is decrypted using a simple XOR operation with the key value 0x36. The decrypted 32-bit shellcode is then executed, leading to the next stage.

Fourth stage

DLL received from the C2

The shellcode employs the same BKDR hashing algorithm mentioned in the third stage to dynamically resolve the APIs. It proceeds to reflectively load an embedded DLL (using the dynamically resolved APIs) from the decrypted C2 data into memory. The DLL contains four exports, DLL entrypoint , load , run , and zidingyixiugaidaochuhanshu . Among these, the DLL entrypoint and load functions are executed.

The load export function copies the observed configuration string in a specific format, reverses the string, and proceeds to parse it. The string is stored in the format |key:value| , where the key represents the configuration attribute and the value represents its corresponding value.

Below is an example:

```
|p1:101.33.117.200|o1:6666|t1:1|p2:101.33.117.200|o2:8888|t2:1|p3:127.0.0.1|o3:80|t3:1|dd:1|cl:1|fz:默认|bb:1.0|bz:2024.1
```

- Keys `p1` | `o1` stores the value C2 IP [Option 1] | C2 port [Option 1].
- Keys `p2` | `o2` stores the value C2 IP [Option 2] | C2 port [Option 2].
- Keys `cl` | `dd` stores the value of how many times the process sleeps, in seconds.

The objective of this stage is to download and execute the final payload. After parsing the C2 configuration and implementing the sleep duration specified in the configuration data, the malware checks if the final payload is already present on the victim host. This is done by opening the registry key `HKEY_CURRENT_USER\Console\0` and querying for the value with the name `d33f351a4aea5e608853d1a56661059`.

If the size of the value is greater than `0xA44`, it indicates that the final payload is already on the victim host. In such cases, the malware proceeds to allocate a `PAGE_EXECUTE_READWRITE` memory section and copies the data from the value of `d33f351a4aea5e608853d1a56661059` into it.

If the final payload does not already exist on the victim host, the malware proceeds to send a DLL named “`(登录模块.dll_bin (Login module.dll_bin))`” to the C2 to download the final payload. The DLL name is encrypted by performing an XOR operation with the same key (`0x36`) used in the third stage.

The response to this request contains the final payload embedded within it. This data is then copied to a `PAGE_EXECUTE_READWRITE` memory section and saved in the registry as a value with the name `d33f351a4aea5e608853d1a56661059` within the key `HKEY_CURRENT_USER\Console\0`.

The embedded DLL is subsequently loaded into memory and executed, serving as the final payload.

Final Payload

The final payload delivered is ValleyRAT, which was initially identified by [Qi An Xin](#) and attributed to the threat actor The Great Thief of Valley, also known as Silver Fox. In this section, we discuss the changes we observed in ValleyRAT, as compared to the [previously documented version](#).

Device fingerprinting

In the latest version of ValleyRAT, the malware developers added new data fields for improved device fingerprinting. The new data collected and sent to the C2 server is bolded in the table below.

Offset	Description	Format (if any)
0x0	Hard coded value (set to 0x06)	
0x2	System IP address	
0x278	Idle time	%d min
0x296	Computer name	
0x2FA	Windows version	

Offset	Description	Format (if any)
0x35E	ntdll.dll version	
0x39A	Number of processors	%d
0x412	HDD & storage device info	HDD:%d WW %d Gb Free %d Gb Mem: %d Gb %sFree%d Gb
0x5A2	GPU info	%s%s %d %d
0x6CE	Foreground window name	
0x8CC	Value of name GROUP of reg key Network/AppEvents (默认 by default)	
0x930	Hardcoded value (set to 1.0)	
0x994	Value of name REMARK of reg key Network/AppEvents (2024.3.6 by default)	
0x9F8	System uptime	运:%s 开:%d.%d.%d %d:%d:%d
0xA5C	RAT architecture (hardcoded X86) followed by victim system architecture.	X86 %s
0xA70	Integrity level to check privilege followed by victim system username.	低/%s (Low), 中/%s (Medium), 高/%s(High), 系 统/%s(System)[one of this values]
0xAD4	Full path of the current process.	
0xCDC	Is camera available	有(have), "X"[one of this values]
0xCE4	Tencent QQ data	
0xEE2	Anti-virus data	

Offset	Description	Format (if any)
0xF46	System language	
0xF86	Monitor resolution	
0x1184	System directory	
0x11E8	System ID	

Table 2: Device fingerprinting information collected by ValleyRAT.

Bot ID generation

The malware developers also made changes to the bot ID generation process. While the hashing algorithm remained the same, the data utilized for the algorithm was modified. The malware now creates an MD5 hash with the following values as arguments:

- computerName
- numberOfProcessors
- ntdllVersion
- systemIP
- integrityLevelfollowedByUsername
- profileGuid

The code sample below shows the algorithm written in Python.

```
import hashlib
def botIDGeneration(computerName, numberOfProcessors, ntdllVersion, systemIP, integrityLevelfollowedByUsername, profileGuid):
    data = computerName.encode("utf-16le")
    data += ntdllVersion.encode("utf-16le")
    data += systemIP.encode("utf-16le")
    data += b'\x20\x00'
    data += numberOfProcessors.encode("utf-16le")
    data += "X86".encode("utf-16le")
    data += integrityLevelfollowedByUsername.encode("utf-16le")
    data += profileGuid.encode("utf-16le")
    data += b'\x00\x00'
    result = hashlib.md5(data).hexdigest()
    print(result)
```

New commands

Finally, the malware developers introduced new commands, which are bolded in the table below.

Opcode	Description
0x1	Load plugin.
0x3	Capture a screenshot of the desktop window and retrieve the name of the foreground window and last input time.
0x4	Capture a screenshot of the entire desktop window.
0x5	Drop and execute a file.
0x6	Download and execute a file from a specified URL using <code>InternetReadFile</code> .
0x7	Set the values of the names <code>GROUP</code> and <code>REMARK</code> in the registry key <code>Network/AppEvents</code> .
0x8	Process filtering using <code>CreateToolhelp32Snapshot</code> .
0xA	Capture a screenshot of the desktop window, where the x and y coordinates of the upper-left corner of the destination rectangle used by the <code>StretchBlt</code> API are determined by C2.
0xB	Clear the Windows event log using the <code>ClearEventLogW</code> function.
0xC	Restart the current process by creating the same process as a child process and subsequently terminating the current process.
0xD	Exit the current process.
0xE	Forced logoff.
0xF	Forced reboot.
0x10	Forced shutdown.
0x11	Change the loading method to a puppet process or an exported function.
0x12	Configuration migration.

Opcode	Description
0x64	Set the value of the name " IpDatespecial " in the registry key HKEY_CURRENT_USER\Console .
0x65	Delete the value named " IpDatespecial " from the registry key HKEY_CURRENT_USER\Console .
0xC9	Retrieve the name of the foreground window and last input time.

Table 3: Commands implemented by ValleyRAT.

Explore more Zscaler blogs

Source: <https://www.zscaler.com/blogs/security-research/technical-analysis-latest-variant-valleyrat>