



# maliciouslife

BY CYBEREASON

[Search](#)[Subscribe](#)

# A Bazar of Tricks: Following Team9's Development Cycles

WRITTEN BY [Cybereason Nocturnus](#)

**Research by:** Daniel Frank, Mary Zhao and Assaf Dahan



- **A New Malware Family:** The Cybereason Nocturnus team is tracking a new Bazar loader and backdoor that first emerged in April 2020 and has evolved continuously since. Bazar can be used to deploy additional malware, ransomware, and ultimately steal sensitive data from organizations.
- **Targeting the US and Europe:** Bazar malware infections are specifically targeting professional services, healthcare, manufacturing, IT, logistics and travel companies across the US and Europe.
- **With Loader and Backdoor Capabilities:** Bazar leverages the Twilio SendGrid email platform and signed loader files to evade traditional security software in conjunction with a fileless backdoor to establish persistence.
- **Under Constant Development:** Over the course of this investigation, it is evident that Bazar is under active development. More recently, the active campaigns have disappeared, but later reappeared with a new version, which indicates the group is under a development cycle.
- **Evasive, Obfuscated Fileless Malware:** This stealthy loader evades detection by abusing the trust of certificate authorities, much like previous Trickbot loaders. This loader, however, uses EmerDNS (.bazar) domains for command and control and is heavily obfuscated. It also uses anti-analysis techniques to thwart automated and manual analysis, and loads the encrypted backdoor solely in memory.
- **A Comeback After Two Months:** After a two month hiatus, a new variant emerged in mid-June that improved on its stealth capabilities. This is similar to the modus operandi of other cybercriminal organizations in general and Trickbot in particular.



generation, they share the same top-level Bazar domain C2. Unlike Trickbot and Anchor, the Bazar loader and backdoor decouple campaign and bot information in bot callbacks. Given these ties and how quickly Bazar is evolving, this may signal the attackers next generation of malware attacks.

## TABLE OF CONTENTS

- [Key Findings](#)
- [Introduction](#)
- [Infection Vector](#)
- [Loader and Backdoor Analysis](#)
- [The Early Development Loader \(Team9\)](#)
- [The Operational Bazar Loader](#)
- [The New Operational Bazar Loader](#)
- [The Early Development Backdoor \(Team9\)](#)
- [The Trickbot Connection](#)
- [Conclusion](#)
- [MITRE ATT&CK Techniques](#)
- [IOCs](#)

## INTRODUCTION

Since April 2020, the Cybereason Nocturnus team has been investigating the emergence of the Bazar malware, a loader and backdoor used to collect data about the infected machine and to deploy additional malware. In this analysis, we show how the Bazar malware is sent via phishing emails that take advantage of the ongoing coronavirus pandemic,



investigation, it is clear that the same infection chain delivers the Bazar loader instead of the usual Trickbot downloader.

The Bazar loader and Bazar backdoor are named after their use of EmerDNS blockchain domains. Using Bazar domains has been trending recently among cybercriminals because they are able to evade takedowns and sinkholing that disrupts botnet communications.

The Bazar loader gives the attacker its initial foothold in the environment, while the Bazar backdoor establishes persistence. Together, the loader and backdoor give threat actors the opportunity to deploy other payloads such as ransomware, and post-exploitation frameworks like CobaltStrike, as well as exfiltrate data and remotely execute commands on infected machines. The Bazar backdoor can lead to disrupted business continuity, data loss, and full compromise, undermining trust in an organization.

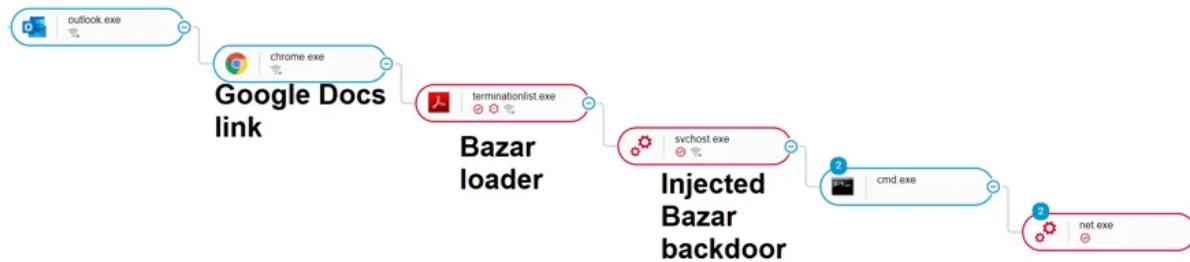
There are several different versions of the Bazar backdoor and its loader, which shows that the malware is under active development. This writeup dissects the Bazar loader and backdoor functionality alongside elements that show its ties to Trickbot collaborations similar to that of Trickbot-Anchor from 2019. Our analysis will focus mainly on the Bazar loader as it is especially evasive given our findings from its recent re-emergence.



Phishing email

The Bazar loader infection chain starts from a phishing email link.

## INFECTION VECTOR



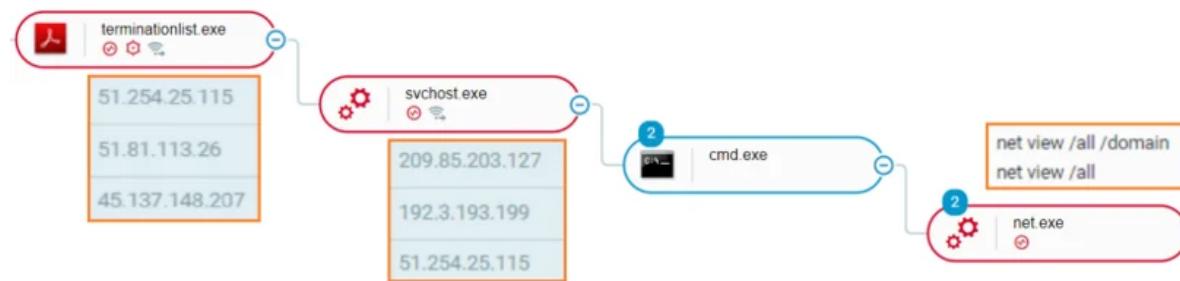


attachments to launch Microsoft Office macros and download Trickbot, this campaign initially infects hosts with the Bazar loader via phishing emails sent using the Sendgrid email marketing platform. These emails contain links to decoy landing pages for document previews hosted in Google Docs.



*Coronavirus phishing email sent via Sendgrid email marketing with Google Docs links.*

Visiting the Google Docs landing page encourages the user to download a file. To convince users to download the files manually, the page states that document preview is not available.



The Bazar loader payload retrieval and net.exe commands post-infection.

The Bazar loader files are dual-extension executable files (such as PreviewReport.DOC.exe) signed with fake certificates such as VB CORPORATE PTY. LTD. This is consistent with the Trickbot group, which notoriously abuses the trust of certificate authorities by using signed



and their effectiveness in evading security products.

signature:"VB CORPORATE PTY"]

835EDF1EC33FF1436D354AA52E2E180E3E8F7500E9D261D1FF26AA6DADDFFC55

Preview\_Employee\_Report.exe  
  🕒 peexe   assembly   overlay   revoked-cert   runtime-modules   signed   direct-cpu-clock-access  
  checks-user-input   64bits   🔗   📍

55D95D9486D77DF6AC79BB25EB8B8778940BAC27021249F779198E05A2E1EDAE

AprilsReport.exe  
  🕒 peexe   assembly   overlay   revoked-cert   runtime-modules   signed   direct-cpu-clock-access  
  checks-user-input   64bits   🔗   📍

4E4F9A467DD041E6A76E2EA5D57B28FE5A3267B251055BF2172D9CE38BEA6B1F

GridCtrlDemo.EXE  
  🕒 peexe   assembly   overlay   revoked-cert   runtime-modules   signed   direct-cpu-clock-access  
  checks-user-input   64bits   🔗   📍

859FA9ACF0B8A989A1634A1EEE309355438B9F6B6F73B69F12D53AC534618C6A

GridCtrlDemo.EXE  
  🕒 peexe   assembly   overlay   revoked-cert   runtime-modules   signed   direct-cpu-clock-access  
  checks-user-input   64bits   🔗   📍

5DBE967BB62FFD60D5410709CB4E102CE8D72299CEA16F9E8F80FCF2A1FF8536

MFCSplne.EXE  
  🕒 peexe   revoked-cert   runtime-modules   signed   overlay   🔗   📍

Trickbot and Bazar loader signed files.

## LOADER AND BACKDOOR ANALYSES

The Cybereason Nocturnus team analyzed both development and operational versions of the Bazar loader and backdoor. To differentiate between the two versions for this writeup, we reserved the name "Team9" for the development versions and the name "Bazar" for the operational versions.

The Team9 loader is examined first; then, we analyze the operational Bazar loader. Finally, we analyze an early development version of the malware, which is the Team9 backdoor. We summarize changes between



Loader variant	Creation date	Mutex	Log files (if any)
Dev Version 1	April 9	n/a	ld_debuglog.txt
Operational Loader	March 27 - April 20	ld_201127	n/a
New Operational Loader	June 12 - June 18	ld_201127	n/a

Loader information

Backdoor variant	Creation date	Mutex	Log Files (if any)
Dev Version 1	April 7-9	MSCTF.[botID]	bd_debuglog.txt
Dev Version 2	April 16-22	{589b7a4a-3776-4e82-8e7d-435471a6c03c} AND {517f1c3d-ffc0-4678-a4c0-6ab759e97501}	dl2.log
Dev Version 2.1	April 17-23	{589b7a4a-3776-4e82-8e7d-435471a6c03c}	bd2.log



## Backdoor information

## THE EARLY DEVELOPMENT LOADER (TEAM9)

Examining a development version of the loader, which contains 'team9 loader' strings, it downloads a XOR-encoded payload from a remote server, then decodes and injects the payload into a target process using process hollowing or process doppelgänging injection techniques.

To download the Bazar backdoor, the loader communicates with a remote server that sends the payload to the infected machine in encrypted format. On first inspection, the payload does not show a valid PE header. Reversing the Team9 loader sample shows a XOR key of the infection date, in the format YYYYMMDD (ISO 8601).

Retrieving the system time to decrypt the payload.

The loop responsible for the byte-by-byte decryption is represented in the image below.

A screenshot of a debugger interface showing assembly code. The code is highlighted with a green box and shows a loop structure. The assembly instructions are as follows:

```
loc_140002DB0:
xor    edx, edx
mov    rax, rdi
div    rcx
movzx eax, [rsp+rdx+480h+Data]
xor    [rdi+rbx], al
inc    rdi
cmp    rdi, r8
jb     short loc_140002DB0
```



with the obfuscated and packed variant. This loader variant creates a simple autorun key at *CurrentVersion\Run*, masqueraded as *BackUp Mgr.*

\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run			
	Name	Type	Data
	(Default)	REG_SZ	(value not set)
	BackUp Mgr	REG_SZ	C:\Users\admin\AppData\Roaming\Microsoft\

The autorun key created by the Team9 loader.

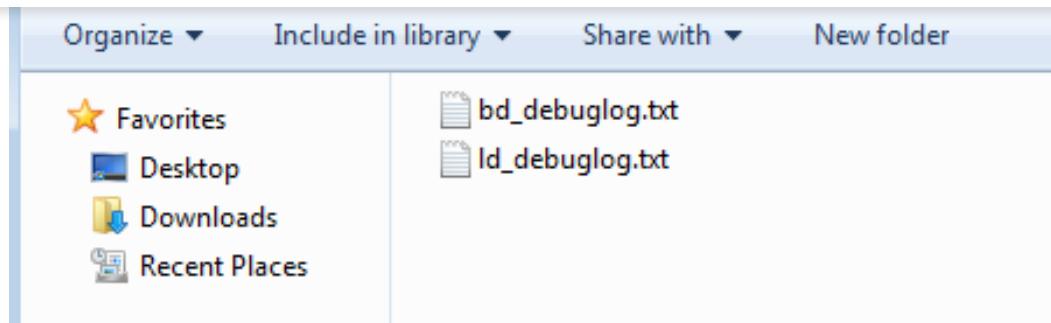
Once the payload is decoded correctly with a proper PE header, it is validated and then injected into memory. The process can be viewed in the malware's logs.

```
d:\development\team9\team9_restart_loader\team9_restart_loader\winmain.cpp:wirMain:60:[-] Installing software.
d:\development\team9\team9_restart_loader\team9_restart_loader\install_utils.cpp:isPlaceOk:123:[-] Checking folder.
d:\development\team9\team9_restart_loader\team9_restart_loader\install_utils.cpp:isPlaceOk:124:[-] filePath: C:\Users\Administrator\Desktop\
d:\development\team9\team9_restart_loader\team9_restart_loader\install_utils.cpp:isPlaceOk:159:[-] software is running from desktop folder.
d:\development\team9\team9_restart_loader\team9_restart_loader\install_utils.cpp:Install:58:[-] Copying software to safe folder.
d:\development\team9\team9_restart_loader\team9_restart_loader\install_utils.cpp:AddToAutorun:15:[-] Adding To Autorun.
d:\development\team9\team9_restart_loader\team9_restart_loader\install_utils.cpp:SelfDelete:176:[-] modulefilePath: C:\users\Administrator\Ap
d:\development\team9\team9_restart_loader\team9_restart_loader\install_utils.cpp:SelfDelete:190:[-] selfdel batch path: C:\users\ADMINI~1\Ap
d:\development\team9\team9_restart_loader\team9_restart_loader\install_utils.cpp:SelfDelete:205:[-] Can't open file. Error: 0
d:\development\team9\team9_restart_loader\team9_restart_loader\install_utils.cpp:SelfDelete:211:[-] Executing batch file for selfdelete.
d:\development\team9\team9_restart_loader\team9_restart_loader\winmain.cpp:wirMain:160:[-] Downloading payload.
d:\development\team9\team9_restart_loader\team9_restart_loader\http_utils.cpp:DownloadFile:22:[-] Download URL: http://bestgame.bazar/api/v108
d:\development\team9\team9_restart_loader\team9_restart_loader\http_utils.cpp:ParseUrl:146:[-] InternetCrackUrlA Failed. Error: 0
d:\development\team9\team9_restart_loader\team9_restart_loader\http_utils.cpp:HttpRequestSend:71:[-] host: bestgame.bazar path: /api/v108
d:\development\team9\team9_restart_loader\team9_restart_loader\http_utils.cpp:HttpRequestSend:81:[-] obtainUserAgentString res: 0
d:\development\team9\team9_restart_loader\team9_restart_loader\http_utils.cpp:HttpRequestSend:88:[-] User-Agent: Mozilla/4.0 (compatible; MS
d:\development\team9\team9_restart_loader\team9_restart_loader\http_utils.cpp:HttpRequestSend:216:[-] bytesAvailable: 898
d:\development\team9\team9_restart_loader\team9_restart_loader\http_utils.cpp:HttpRequestSend:216:[-] bytesAvailable: 2048
d:\development\team9\team9_restart_loader\team9_restart_loader\http_utils.cpp:HttpRequestSend:216:[-] bytesAvailable: 0
d:\development\team9\team9_restart_loader\team9_restart_loader\winmain.cpp:wirMain:189:[-] Payload size: 182272
d:\development\team9\team9_restart_loader\team9_restart_loader\winmain.cpp:wirMain:221:[-] Payload is Ok.
```

Contents of the log file (*ld\_debug.txt*) show Bazar loader infection activity.

Debug strings show the Bazar loader execution and payload retrieval status in a log file "ld\_debuglog" indicating PE file signature verification and self-deletion capabilities.

This variant places the debug logs in the hardcoded 'admin' user folder.



Bazar loader and backdoor debug logs.

## THE OPERATIONAL BAZAR LOADER

In the obfuscated and packed version of the loader, an uncommon API call is used to facilitate code injection. As seen in the image below, the loader uses *VirtualAllocExNuma* to allocate new memory and store the returned base address. The beginning of an obfuscated shellcode is copied to this address after being decrypted using an RC4 algorithm. In addition to the shellcode an additional PE can be seen in memory.



```

mov    rcx, rax      ; hModule
call   cs:GetProcAddress
mov    cs:VirtualAllocExNuma, rax
mov    [rsp+78h+size], ebp
lea    rcx, [rsp+78h+key_arg]
mov    rax, cs:key_part1 ; 'mElhVkuJ'
mov    [rcx], rax
mov    rax, cs:key_part_2 ; 'dpW7CRO'
mov    [rcx+8], rax
call   cs:GetCurrentProcess
mov    rcx, rax
mov    [rsp+78h+var_50], ebp
mov    [rsp+78h+var_58], 40h ; '@'
xor   edx, edx
mov    r9d, 3000h
mov    r8d, 27A12h
call   cs:VirtualAllocExNuma
rsi, rax
mov    r8d, 27A12h      ; Size
lea    rdx, encrypted_shellcode_and_mz ; Src
mov    rcx, rax          ; void *
call   memmove
mov    [rsp+78h+size], 27A12h
lea    r9, [rsp+78h+size]
mov    r8, rsi
mov    edx, 1
lea    rcx, [rsp+78h+key_arg]
call   decrypt_shellcode_and_mz
test  al, al
jnz   short loc_4057F9

```

Memory allocation and call to shellcode decryption.

The Bazar loader also stores an RSA2 key that is used to open the RC4 key.

07	02	00	00	00	A4	00	00	52	53	41	32	00	02	00	00	.....¤..RSA2....
01	00	00	00	AB	EF	FA	C6	7D	E8	DE	FB	68	38	09	92	.....«iÚÆ}èPÙh8..
D9	42	7E	6B	89	9E	21	D7	52	1C	99	3C	17	48	4E	3A	ÜB~k..!xR..<.HN:
44	02	F2	FA	74	57	DA	E4	D3	C0	35	67	FA	6E	DF	78	D.òútwÙäóÀ5gúnßx
4C	75	35	1C	A0	74	49	E3	20	13	71	35	65	DF	12	20	Lu5. tIä .q5eß.
F5	F5	F5	C1	ED	5C	91	36	75	B0	A9	9C	04	DB	0C	8C	öööÄí\.6u°@..0..
BF	99	75	13	7E	87	80	4B	71	94	B8	00	A0	7D	B7	53	ż.u.~..Kq. . } .S
DD	20	63	EE	F7	83	41	FE	16	A7	6E	DF	21	7D	76	C0	Ý cî÷.Ap.Şnß!}vÁ
85	D5	65	7F	00	23	57	45	52	02	9D	EA	69	AC	1F	FD	.Óe..#WER..êi~.ý
3F	8C	4A	D0	01	00	00	00	00	00	00	00	00	00	00	00	?..JØ.....

RSA2 BLOB as seen in the loader's memory.

Looking at the code of the 'decrypt\_shellcode\_and\_mz' function, we see it is very similar to the one being used in an earlier Trickbot variant and



```

    && !CryptAcquireContextW(&phProv, 0i64, 0i64, 1u, 8u)
    && !CryptAcquireContextW(&phProv, 0i64, 0i64, 1u, 8u) )
{
    goto LABEL_18;
}
hPubKey = 0i64;
if ( !CryptImportKey(phProv, &rsa_blob, 0x134u, 0i64, 0, &hPubKey) )
    goto LABEL_18;
v9 = 0i64;
if ( (int)v4 > 0 )
{
    v10 = (BYTE *) (v4 + a1 - 1);
    do
    {
        v11 = *v10;
        ++v9;
        --v10;
        rc4_blob[v9 + 11] = v11;
    }
    while ( v9 < v4 );
}
rc4_blob[v4 + 12] = 0;
if ( (int)v4 + 1 < 62i64 )
{
    LOBYTE(v8) = 1;
    memset(&rc4_blob[(int)v4 + 13], v8, 62i64 - ((int)v4 + 1));
}
hKey[0] = 0i64;
if ( CryptImportKey(phProv, rc4_blob, 0x4Cu, hPubKey, 0, hKey) )
    result = CryptEncrypt(hKey[0], 0i64, 1, 0, a3, a4, *a4);

```

The shellcode decryption routine.

After the RSA2 key is imported from the key BLOB, the RC4 key is loaded into the RC4 BLOB. It is reversed, since it defaults to the little-endian format, and is finally appended with a trailing zero byte, which is an essential part of the key.

01 02 00 00	01 68 00 00	00 A4 00 00	00 64 70 57	.....h.....dpw
37 43 52 4F	6D 45 6C 68	56 6B 75 4A	00 01 01 01	7CROMElhvkuJ....
01 01 01 01	01 01 01 01	01 01 01 01	01 01 01 01	.....
01 01 01 01	01 01 01 01	01 01 01 01	01 01 01 01	.....
01 01 01 01	01 01 01 01	01 01 02 00	00 00 00 00	.....
BC 52 1A E8	1C 66 3A 7E	C9 BD 25 FB	7E 63 4F E9	%R.é.f:~É%0~cOé
49 09 DF B5	63 88 F3 7F	34 8F 8B EE	9B 78 68 5F	I.ßuc.ó.4..í.xh_
E1 AA AE 30	3E AD 59 BD	F8 78 C1 69	75 6D 37 C4	á@0>.Y%o{Áium7Ä
26 21 83 9F	98 08 BE 6B	A7 AD C3 80	44 65 CA BD	&!....%k§.Ä.De%

The RC4 BLOB with the loaded key.



```

OD 44 03 D8 80 7B FF 00 75 ED 41 8D 04 13 3B C6 .D.Ø€(ÿ.uiA...;È
74 0D FF C1 41 3B 4A 18 72 D1 E9 5B FF FF FF 41 t.ÿÁA;J.rÑé[ÿÿÿA
8B 42 24 03 C9 49 03 C0 0F B7 14 01 41 8B 4A 1C <BS.ÉI.À. ...A< J.
49 03 C8 8B 04 91 49 03 C0 EB 02 33 C0 48 8B 5C I.È<. 'I.Àë.3ÀH<\ 
24 20 48 8B 74 24 28 48 83 C4 10 5F C3 4D 5A 90 $ H<t$(HfÄ._ÄMZ.
00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 .....YY...
00 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00 00 .....@.....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
00 00 00 00 00 00 00 00 00 08 01 00 00 0E 1F BA ..... .
0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 69 73 20 ..'.í!..Lí!This
70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F 74 20 62 program cannot b
65 20 72 75 6E 20 69 6E 20 44 4F 53 20 6D 6F 64 e run in DOS mod
65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 BF F8 2B e....$. ....¿ø+
E6 FB 99 45 B5 FB 99 45 B5 FB 99 45 B5 4F 05 B4 æù"Euû"Euû"EuO. '
B5 FE 99 45 B5 4F 05 B6 B5 80 99 45 B5 4F 05 B7 ùp"EuO. Ùp"EuO. -
B5 F6 99 45 B5 B1 FC 46 B4 F1 99 45 B5 B1 FC 41 ùp"Eu+üF' ñ"Eu+üA
B4 E8 99 45 B5 B1 FC 40 B4 D1 99 45 B5 F2 E1 C6 'è"Eu+ü@' Ñ"EuòáÈ
B5 F9 99 45 B5 F2 E1 C2 B5 F9 99 45 B5 F2 E1 D6 ùù"EuòáÀùù"EuòáÖ
B5 FE 99 45 B5 FB 99 44 B5 99 99 45 B5 EB FF 4C ùp"Euû"Du"ù"EuëýL
B4 F4 99 45 B5 EB FF BA B5 FA 99 45 B5 EB FF 47 'ò"Euëý"ùù"EuëýG
B4 FA 99 45 B5 52 69 63 68 FB 99 45 B5 00 00 00 'ú"EuRichù"Eu...
00 00 00 00 00 50 45 00 00 64 86 06 00 3A 86 91 .....PE..dt...:t'
5E 00 00 00 00 00 00 F0 00 22 00 0B 02 0E ^.....ð.".....

```

The decrypted shellcode and PE.

Copied to the previously allocated memory, this code deobfuscates several essential API calls at runtime, such as *LoadLibraryA*, *GetProcAddress*, *VirtualAlloc* and *VirtualProtect*, all of which will be used to resolve APIs and allocate memory to run the additional PE.

E8 A4040000	<code>call 1EB0518</code>	<code>LoadLibraryA</code>
B9 49F70278	<code>mov ecx,7802F749</code>	
4C:8BE8	<code>mov r13,rax</code>	
E8 97040000	<code>call 1EB0518</code>	<code>GetProcAddress</code>
B9 58A453E5	<code>mov ecx,E553A458</code>	
48:894424 20	<code>mov qword ptr ss:[rsp+20],rax</code>	
E8 88040000	<code>call 1EB0518</code>	<code>VirtualAlloc</code>
B9 10E18AC3	<code>mov ecx,C38AE110</code>	
48:8BF0	<code>mov rsi,rax</code>	
E8 7B040000	<code>call 1EB0518</code>	<code>VirtualProtect</code>
B9 AFB15C94	<code>mov ecx,945CB1AF</code>	

API resolving by the shellcode loader.

The code loads more APIs to the soon-to-be-executed PE before finally



41:0FB717	movzx edx,word ptr ds:[r15]	edx:"CreateFileW"
42:8B8C20 88000000	mov ecx,dword ptr ds:[rax+r12+88]	
42:8B4421 10	mov eax,dword ptr ds:[rcx+r12+10]	
42:8B4C21 1C	mov ecx,dword ptr ds:[rcx+r12+1C]	
48:2BD0	sub rdx,rax	
49:03CC	add rcx,r12	
8B0491	mov eax,dword ptr ds:[rcx+rdx*4]	
49:03C4	add rax,r12	
EB 0F	jmp 600284	
49:8B16	mov rdx,qword ptr ds:[r14]	rdx:"CreateFileW"
49:8BC0	mov rcx,r12	
48:83C2 02	add rdx,2	rdx:"CreateFileW"
48:03D6	add rdx,rsi	rdx:"CreateFileW"
FFD5	call rbp	
49:8906	mov qword ptr ds:[r14],rax	
49:83C6 08	add r14,8	
49:83C7 08	add r15,8	
49:833E 00	cmp qword ptr ds:[r14],0	

Resolving APIs for the PE by the shellcode loader.

Stepping into the loaded PE, Bazar loader tries to avoid targeting Russian users by checking if the Russian language is installed on the infected machine. It calls *setlocale*, deobfuscating the "Russia" string by adding 0xf4 to each character, and finally resolving and calling *StrStrA* to check if "Russia" is a substring of the current locale. If so, the loader terminates. The Bazar Backdoor repeats this step as well.

C745 85 5E817F7F	mov dword ptr ss:[rbp-7B],7F7F815E	
66:C745 89 756D	mov word ptr ss:[rbp-77],6D75	
44:8875 88	mov byte ptr ss:[rbp-75],r14b	
804405 85 F4	add byte ptr ss:[rbp+rax-7B],F4	
49:03C7	add rax,r15	
48:83F8 06	cmp rax,6	
72 F2	jb 1FE49BA	
BA 13000000	mov edx,13	edx:"Russia"
41:B9 14020000	mov r9d,214	
41:B8 E6767C2A	mov r8d,2A7C76E6	
E8 F2F7FFFF	call 1FE41D0	
48:85C0	test rax,rax	
74 0F	je 1FE49F2	
48:8D55 85	lea rdx,qword ptr ss:[rbp-7B]	
48:8D0D 42340200	lea rcx,qword ptr ds:[2007E30]	rcx:"English_United States.1252"
FFD0	call rax	StrStrA

Checking for Russian language to determine if it should execute.

In general, the PE is highly obfuscated. Dedicated methods resolve additional strings and API calls at runtime, rendering the PE even more difficult to analyze. Below is an example of the method responsible for



```

66:0F6F05 6E220200 movdqa xmm0,xmmword ptr ds:[2003520]
F3:0F7F45 C7 movdqu xmmword ptr ss:[rbp-39],xmm0
49:8BCE mov rcx,r14
8A45 C7 mov al,byte ptr ss:[rbp-39]
30440D C8 xor byte ptr ss:[rbp+r14-38],al
49:03CF add rcx,r15
48:83F9 0E cmp rcx,E
72 F0 jb 1FE12BA
44:8875 D6 mov byte ptr ss:[rbp-2A],r14b
48:8D55 C8 lea rdx,qword ptr ss:[rbp-38]
48:8D4D D7 lea rcx,qword ptr ss:[rbp-29] [rbp-29]:"bestgame.bazar"

```

Deobfuscating .bazar domains.

A mutex name is deobfuscated and then copied before being passed to *CreateMutexExA* with the name "Id\_201127".

```

41:B9 80000000 mov r9d,80
41:8BD7 mov edx,r15d
E8 B7FAFFFF call 1FE41D0
48:85C0 test rax,rax
74 0B je 1FE4729
48:8D5424 7B lea rdx,qword ptr ss:[rsp+7B]
48:8D4D A0 lea rcx,qword ptr ss:[rbp-60]
FFD0 call rax
45:33C9 xor r9d,r9d
48:8D55 A0 lea rdx,qword ptr ss:[rbp-60]
45:33C0 xor r8d,r8d
33C9 xor ecx,ecx
FF15 753B0200 call qword ptr ds:[<&CreateMutexExA>] edx:"Id_201127"
                                                               resolve lstrcpy
                                                               rax:"Id_201127"
                                                               call lstrcpy

```

Mutex creation

Once the Bazar loader downloads its payload, the Bazar backdoor, it is decrypted using the same method as the aforementioned *Team9* variant.



```
loc_1FE2166:  
mov    cl, [rsp+rbx+systemtime]  
lea    eax, [rbx+1]  
xor    [rdi], cl  
inc    rdi  
cmp    eax, r14d  
sbb    ebx, ebx  
and    ebx, eax  
sub    rdx, 1  
jnz    short loc_1FE2166
```

Decrypting the downloaded payload.

Finally, the loader validates the PE header for successful decryption, then it continues to the next step, which is code injection by process hollowing.

System time retrieval, decryption, and header check of the downloaded payload.

The loader tries three different processes: *svchost*, *explorer*, and *cmd*, similar to the functionality in the development version.

After the code is successfully injected into one of the above processes, the loader uses several methods to autorun from the victim's machine. This implies that the code has not yet been finalized.

Bazar loader making sure it will autorun at any cost.

First, the loader creates a scheduled task masquerading under the name *StartAd - Ad as in Adobe*. Other samples use a decoy Adobe icon with a double extension *.PDF.exe*, similar to the MS Word variant being analyzed here.



```
v 74 44          | je 1FE8721  
41:8BCF         | mov ecx,r15d  
F0:0FC14B 10   | lock xadd dword ptr ds:[rbx+10],ecx  
41:03CF         | add ecx,r15d  
v 75 33          | jne 1FE871D
```

Creation of the scheduled task using taskschd.dll.

The author is also set as *Adobe* for further deception.

The created task as seen in the Task Scheduler.

After setting up the scheduled task, the Bazar loader uses *RegSetValueExA* to write itself to *HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon*. By doing so, the loader is able to execute on every system logon.

Userinit REG\_SZ C:\Windows\system32\userinit.exe,C:\Users\Administrator\Desktop\PreviewReport.DOC.exe

Writing the malware to autorun from userinit.

The Bazar loader will create another autorun entry by writing an *adobe.lnk* shortcut in the Windows Start menu Startup folder.

Writing the Bazar loader to the startup folder.

Finally, if the autorun overkill process was not enough, the malware grabs the user's desktop using the *SHGetSpecialFolderPathW* API call, and makes the shortcuts point to the loader itself. It opens each shortcut location, renaming the target by prefixing the application's name with an underscore, ultimately renaming itself as the original application, copied to the destination folder.



For example, the screenshot above shows that `_firefox.exe` is the original application, while `firefox.exe` is actually a copy of the Bazar loader. This is confirmed after retrieving the files' hashes.

Hashing both malicious loader copy and legitimate Firefox applications.

Another small binary file is created in the folder with a `.bin` extension, containing more encrypted data.

## THE NEW OPERATIONAL BAZAR LOADER

A new version of the Bazar loader emerged at the beginning of June 2020. The files submitted to VirusTotal share the same fake certificate: "RESURS-RM OOO". While some functionality remains similar to the older operational variant (such as the mutex, the downloaded payload decryption routine, the persistence mechanism etc.), there are some new features in this new variant.

One noticeable feature is the evasive API-Hammering technique, that was also seen recently in a new Trickbot variant. In this case, the usage of 1550 calls to `printf` is intended to overload sandbox analysis with junk data and delay execution, since it logs API calls.

Bazar loader's API-Hammering technique.

Another noticeable difference in the new variant is the change to the initial shellcode decryption routine, though it uses the familiar `VirtualAllocExNuma` routine.



following key.

The key used for the shellcode decryption.

Once the code is decrypted, it is clear that there are actually two payloads inside of it. The first payload serves as a loader for the second DLL payload.

The first PE loads the second one with the export function "StartFunc".

Offset *0x180004000* holds the second DLL.

The second DLL.

Once loaded, the second DLL's *StartFunc* starts a loop by calling *GetMessageA* to retrieve Windows messages and runs the main activity method.

*StartFunc* main activity method.

Another interesting finding is that Bazar Loader has now implemented a Domain Generation Algorithm using the current date as a seed. At the moment, it seems more of a backup, since in monitored live cases the IPs were contacted directly.

Bazar Loader's DGA implementation.



Generated Bazar domains.

Other more minor (but significant for detection) changes include:

- Connecting to the C2 using only HTTPS
- User-Agent name was changed to *dbcutfwq* or *user\_agent*
- A new cookie: group=1
- *\_lyrt* suffix that was used to check the malware's presence on the machine now changed to *\_fgqw*

## THE EARLY DEVELOPMENT BACKDOOR (TEAM9)

The Cybereason Nocturnus team has identified three versions of this backdoor since early April this year. Their modus operandi does not differ drastically and can be distinguished by their mutexes and obfuscation level.

Data collected from the infected machine is hashed using the MD5 algorithm set in the *CryptCreateHash* API call by setting the ALG\_ID to 0x8003, and then appended to the mutex name.

Gathering and hashing data about the infected machine.

After successfully gathering the data, the Bazar backdoor connects to the C2 server. If the connection fails, it continues to retry.

Another interesting aspect of this version is how it uses a local address to fetch the data from the server. Given that this is an early dev version, the author may be using this method for test purposes.



backdoor parses the command received in the HTTP response. Each char of the command is XORed with the next char in the generated MD5 string.

XORing the command retrieved from the C2 with the machine identifier hash.

After checking and parsing the XORed data, the backdoor then logs and executes the retrieved command according to the following switch case.

Switch case for the commands received from the C2 server.

As seen in the above image, the Bazar backdoor can handle quite a few commands. This next section focuses on case 1, which retrieves various pieces of additional information on the infected machine.

After receiving the value 1 from the C2 server and parsing the response, the value is mapped to the relevant method for execution.

The methods and mapped values as seen in memory.

The corresponding method to the value 1 is *0x3fab15b0* in this instance. This method collects additional data from the infected machine, such as its public IP address, computer name, and the installed Windows version.

Gathering additional information about the infected machine.

It then performs a WMI query to retrieve information about the antivirus engine installed on the machine.



*Windows\CurrentVersion\Uninstall* registry key.

Querying the installed programs on the machine.

Finally, the loader spawns *cmd.exe* to perform a series of reconnaissance commands to obtain information about the network and domain.

*cmd.exe* running net and nltest tools.

Because the malware is a development version, most of the above data is well-documented in its logs.

Team9 backdoor logs.

Subsequent network communications use a bot ID hash format reminiscent of the client ID used in Anchor campaigns from 2019, an MD5 hash value.

As seen in previous Anchor infections, Anchor's unique identifier generation follows this pattern:

**[Machine\_NAME]\_[Windows\_Version].[Client\_ID]**

After a machine is infected with Anchor, it uses openNIC resolvers to resolve a Bazar domain such as `toexample[dot]bazar`. It then sends bot callbacks with the following information to the remote server in the format shown below:

**[campaign]/[Machine\_NAME]\_[Windows\_Version].[Client\_ID]/[switch]**



The Bazar bot ID is an MD5 hash comprised of host information, including:

- [creation date of %WINDIR% in ASCII]
- [creation date of %WINDIR%\system32 in ASCII].
- [NETBIOS\_Name]
- [%SYSTEMDRIVE% serial number])

Bazar backdoor communications follow a pattern of the botID and numeric command switch.

**[botID]/[switch]**

Backdoor callbacks from the infected host to the Bazar domain use the botID and command switch '2' when waiting to receive a new task.

Network communication from infected host to the .bazar domain with a unique botID.

The Bazar backdoor sends a 'group' identifier to the remote server along with the botID and the switch to send data or receive commands. As of May 2020, there were two hardcoded groups. These backdoors are associated with cookie group strings "two" and "five". Meanwhile, the new loader is associated with the cookie group string, "1".

Bazar backdoor "group" identifier sent via HTTP request "cookie" parameter.

While the URI string has changed from Trickbot and Anchor variants, the phishing tactics and use of post-infection reconnaissance commands remains the same. In the Bazar backdoor, the tag (or gtag) used to identify



With Bazar, the infected machine name and Trickbot campaign identifier are no longer sent in the same HTTP requests. Instead, the '/api/v{rand}' URI is sent to retrieve the backdoor from cloud hosted servers after the loader executes. Backdoor communications between the C2 server and the client occur to the .bazar domain using the botID assigned to the infected host.

The decoupling of campaign tag and client machine name from the Bazar C2 server is specific to this backdoor. Because bot communications are often quickly terminated after infections are discovered, removing the campaign and client machine name from URIs results in reduced downtime, lowering the need to re-infect a machine.

## THE TRICKBOT CONNECTION

As we previously stated, the Bazar loader and Bazar backdoor show ties to Trickbot and Anchor malware with signed loaders. Similarities between the three include:

- using revoked certificates to sign malware
- domain reuse (e.g. machunion[.]com and bakedbuns[.]com)
- Almost identical decryption routines in the Bazar and Trickbot loaders, including the usage of the same WinAPIs, custom RC4 implementation and the usage of the API-Hammering in the latest loader variant, which is found also in Trickbot.
- backdoor command-and-control using .bazar domains

The fact that this malware does not infect machines with Russian language support offers a clue to its origins and intended targets.



certificate issued to subject "VB CORPORATE PTY. LTD." in January 2020. Anchor campaigns from December also used signed Trickbot loader files with filenames related to preview documents. The current revoked certificate used in the new loader variant is issued by "RESURS-RM OOO".

In addition, similar phishing email tactics, Google Drive decoy previews, signed malware, and deceptive file icon use were observed in both of these campaigns. We observed reuse of likely compromised domains to host Bazar loaders that previously served Trickbot loaders. For example, the domain ruths-brownies[dot]com was used in a Trickbot campaign in January and hosted Bazar loaders in April 2020.

The Bazar malware has a new command-and-control pattern and botID that differs from Trickbot and Anchor, yet retains historical indicators of both malware families. Finally, the use of Emercoin (.bazar) domains were observed in Trickbot infections delivering Anchor from December 2019.

## CONCLUSION

In this writeup, we associate the Bazar loader and Bazar backdoor with the threat actors behind Trickbot and our previous research on Anchor and Trickbot from December 2019. Based on our investigation, Cybereason estimates that the new malware family is the latest sophisticated tool in Trickbot gang's arsenal, that so far has been selectively observed on a handful of high-value targets.

The Bazar malware is focused on evasion, stealth, and persistence. The malware authors are actively testing a few versions of their malware, trying to obfuscate the code as much as possible, and hiding the final payload while executing it in the context of another process. To further



Post-infection, the malware gives threat actors a variety of command and code execution options, along with built-in file upload and self-deletion capabilities. This variety allows attackers to be dynamic while exfiltrating data, installing another payload on the targeted machine, or spreading further on the network. In general, having more options ensures the threat actors can adjust to changes in their goals or victim's environment.

The use of blockchain domains distinguishes the Bazar loader and Bazar backdoor as part of a family of threats that rely on alternate domain name systems such as EmerDNS domains. As we reported in Dropping The Anchor in December 2019, these alternate domain name systems have also been used in Trickbot Anchor campaigns. These systems provide bot infrastructure with protection from censorship and resilience to takedowns, making them invaluable for threat actors.

The emergence of the first malware variants in April 2020 was followed by an almost 2-months long hiatus, until a new variant was discovered in June 2020. Our research, which covers the evolution of the Bazar malware family, clearly shows that the threat actor took time to re-examine and improve their code, making the malware stealthier. Bazar's authors changed some of the most detectable characteristics of the previous variant, such as previously hardcoded strings, and modification of the known shellcode decryption routine, similar to what was previously observed in recent Trickbot variants.

Although this malware is still in development stages, Cybereason estimates that its latest improvements and resurfacing can indicate the rise of a new formidable threat once fully ready for production.

## MITRE ATT&CK TECHNIQUES



		Locati on				
<u>Execution</u> <u>Through</u> <u>API</u>	<u>Startup</u> <u>Items</u>	<u>Startu p</u> <u>Items</u>	<u>Deobfusca te /</u> <u>Decode</u> <u>Files or</u> <u>Information</u>	<u>Account</u> <u>Discovery</u>	<u>Data</u> <u>Encryp ted</u>	<u>Commonly</u> <u>Used Port</u>
	<u>Registry</u> <u>Run Keys</u> <u>/ Startup</u> <u>Folder</u>	<u>Proce ss</u> <u>Injecti on</u>	<u>Masquerad ing</u>	<u>Application</u> <u>Window</u> <u>Discovery</u>		<u>Remote File</u> <u>Copy</u>
			<u>Modify</u> <u>Registry</u>	<u>File and</u> <u>Directory</u> <u>Discovery</u>		<u>Standard</u> <u>Application</u> <u>Layer</u> <u>Protocol</u>
			<u>Obfuscate</u> <u>d Files or</u> <u>Information</u>	<u>Process</u> <u>Discovery</u>		<u>Standard</u> <u>Cryptograp</u> <u>hic Protocol</u>
			<u>Process</u> <u>Doppelgän ging</u>	<u>Query</u> <u>Registry</u>		<u>Standard</u> <u>Non-</u> <u>Application</u> <u>Layer</u> <u>Protocol</u>
			<u>Process</u> <u>Hollowing</u>	<u>Remote</u> <u>System</u> <u>Discovery</u>		



				<u>Discovery</u>		
				<u>System</u> <u>Information</u> <u>Discovery</u>		
				<u>System</u> <u>Time</u> <u>Discovery</u>		
				<u>System</u> <u>Owner /</u> <u>User</u> <u>Discovery</u>		

## INDICATORS OF COMPROMISE

[Click here to download this campaign's IOCs \(PDF\)](#)

### SHARE



### ABOUT THE AUTHOR

Cybereason Nocturnus



The Cybereason Nocturnus Team has brought the world's brightest minds from the military, government intelligence, and



vulnerabilities. The Cybereason Nocturnus Team was the first to release a vaccination for the 2017 NotPetya and Bad Rabbit cyberattacks.

All Posts by Cybereason Nocturnus →

## Related Posts



### THREAT ANALYSIS REPORT: From Shathak Emails to the Conti Ransomware

The ITG23 group is partnering with the TA551 (Shatak) threat group to distribute ITG23's TrickBot and BazarBackdoor malware which attackers use to deploy Conti ransomware on compromised systems...



### Unboxing Snake - Python Infostealer Lurking Through Messaging Services

In this Threat Analysis Report, Cybereason Security Services dives into the Python Infostealer, delivered via GitHub and GitLab, that ultimately exfiltrates credentials via Telegram Bot API or other well known platforms.

Search



SUBSCRIBE



## RECENT POSTS

**Unlocking the Potential of AI in Cybersecurity:  
Embracing the Future and Its Complexities**

**Malicious Life Podcast: Operation Snow  
White, Part 2**

**THREAT ANALYSIS: Beast Ransomware**

## CATEGORIES

Research

Podcasts

Webinars

Resources

Videos

News

[All Posts](#)



# Never miss a blog

Get the latest research, expert insights, and security industry news.

[Subscribe](#)



## About

[who we are](#)

[careers](#)

[contact](#)

## Resources

[blog](#)

[case studies](#)

[webinars](#)

[white papers](#)

## Platform

[overview](#)

[endpoint](#)

[protection](#)

[edr](#)

[mdr](#)

©Cybereason 2024. All Rights Reserved.

[Terms of Use](#) | [Privacy Notice](#) |  
[Do Not Sell](#) | [Security](#)

