

# Smoke Loader malware improves after Microsoft spoils its Campaign

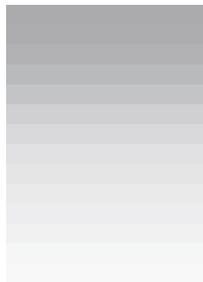
[spamhaus.org/news/article/774/smoke-loader-improves-encryption-after-microsoft-spoils-its-campaign](https://spamhaus.org/news/article/774/smoke-loader-improves-encryption-after-microsoft-spoils-its-campaign)

[Tweet](#) [Follow](#)  
[@spamhaus](#)

Smoke Loader malware improves after Microsoft spoils its Campaign

2018-04-16 15:15:55 UTC | by Spamhaus Malware Labs | Category: [malware](#)

## Recent News Articles



[Spamhaus Botnet Threat Update: Q4-2021](#)

[SERVICE UPDATE | Spamhaus DNSBL users who query via Cloudflare DNS need to make changes to email set-up](#)

[Spamhaus Botnet Threat Update: Q3-2021](#)

Early this year, in March 2018, Microsoft' Windows Defender Research Team in Redmond published some [interesting insights](#) into a massive malware campaign distributing a dropper/loader called Smoke Loader (also known as Dofail). The main purpose of the documented campaign was to distribute a coin miner payload that is using infected machines to mine crypto currencies. Within 12 hours, Windows Defender recorded more than 400,000 instances, but could deploy appropriate countermeasures on computers running Windows within seconds. As further analysis from Spamhaus Malware Labs revealed, these countermeasures did not stay unattended by the malware authors behind Smoke Loader.

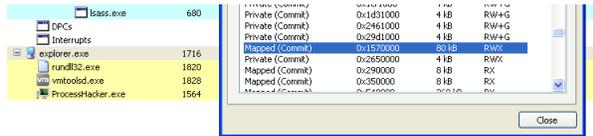
Apparently, as a reaction on Microsoft' countermeasures, the malware authors behind Smoke Loader made some significant code changes in order to bypass Windows Defender and other Antivirus software. These code changes include:

- Change in the infection techniques
- Introduction of 64bit payload

## Anti-VM and Anti-Analysis techniques in the packer

What stares out with Smoke Loader is that the packer and the main executable (unpacked payload) is related to each other. It is necessary for the unpacked payload to be loaded by the packer in order to run. In addition, the unpacked code checks for certain markers created by the packer in order to run. When Smoke Loader gets executed in a sandbox (for example a virtual environment), the sample fails to start. The reason for this are Anti-VM and Anti-Analysis techniques that Smoke Loader implemented in the code recently. An initial examination under IDA reveals that the code is obfuscated with *jump chains* whose sole purpose is to make the static analysis harder.





### Smoke Loader injecting into explorer.exe

The following code change highlights that the final payload is supposed to be run as thread instead of a separate process.

```

lea    eax, [ebp-4]
push  eax
push  ebx
push  0Dh
push  2711h
call  sub_A71EBA
push  eax
xor   eax, eax
call  sub_A7317C
push  ebx
call  ExitProcess
  
```

Previous version (process based)

```

push  MainMutexHandle
call  CloseHandle
push  ebx
call  ExitThread
  
```

Recent version (thread based)

The packer creates a shared file map which contains various information on the initial infection, such as the packed binary. This file map is later being used by the executing thread.

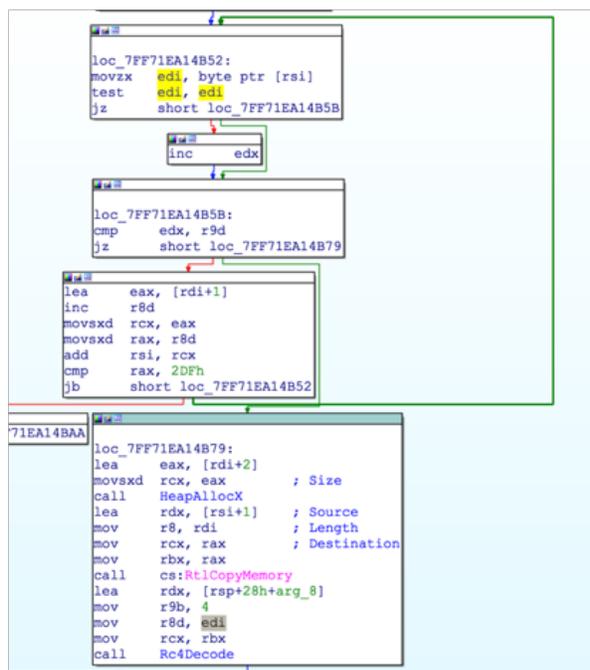
Type	Name	Handle
Desktop	\Default	0x20
Directory	\Windows	0x8
Directory	\Windows\Objects	0x14
Directory	\BaseNamedObjects	0x34
Event	\BaseNamedObjects\{507028746697F055}	0x4c
File	C:\Documents and Settings\Administrator\Desktop\API Call Logger	0xc
File	C:\WINDOWS\WinSxS\x-wwww-3544ce83-Microsoft.Windows.Common-Controls_6595b6414f4f1df_6.0.2600.5512_x-wwww-3544ce83	0x40
Key	HKLM\SYSTEM\ControlSet001\Services\DiskEnum	0x3c
Key	HKLM\SYSTEM\ControlSet001\Services\DiskEnum	0x3c
KeyedEvent	\KernelObjects\CritSecOutOfMemoryEvent	0x4
Section	\BaseNamedObjects\{507028746697F055}	0x8
Semaphore	\BaseNamedObjects\{A4F1A32-A30-11D1-8C6B-0A0C90312E1}	0x30
Token	RAA8HD-03927DE\Administrator: 0xfcd1	0x44
WindowStation	\Windows\WindowStations\WinSta0	0x1c
WindowStation	\Windows\WindowStations\WinSta0	0x24

Shared file map

The name of the shared file map is generated from VolumeSerialNumber of root drive of the infected machine. This shared file map can be used as an indicator of compromise (IOC).

## Additional changes in the code

While the previously string encoding algorithm used by Smoke Loader was based on xor, the most recent version includes an RC4 based string encryption as highlighted on the screenshot above.



### RC4 based string decryption

The following IDA python script can help with static decoding of Smoke Loader: [download](#)

In earlier versions of Smoke Loader, the botnet controller domain names (C&C) were encoded using an algorithm that was based on a simple xor subtraction:

```
def Decodec2(data):
    XorKey = struct.unpack("<B", data[0])[0]
    dst = array.array("B")
    base = data[5:]
    PackLen = struct.unpack("<B", data[4])[0]
    print PackLen
    for i in range(0, PackLen - 1, 2):
        #print chr( ( ( ord (base[i]) ^ XorKey) & 0xff) -
        ((ord(base[i + 1]) ^ XorKey) & 0xff) & 0xff),
        dst.append(( ( ord (base[i]) ^ XorKey) & 0xff) -
        ((ord(base[i + 1]) ^ XorKey) & 0xff) & 0xff)
    return dst.tostring()
```

The most recent version of Smoke Loader has been modified by the authors to make use of a more complex encoding scheme which is based on multiple operations:

```

def swap32(x):
    return (((x << 24) & 0xFF000000) |
            ((x << 8) & 0x00FF0000) |
            ((x >> 8) & 0x0000FF00) |
            ((x >> 24) & 0x000000FF))
def Decodec2(buf):
    BufLen = struct.unpack("<B", buf[0])[0]
    print "[ ] Buf len = %d" % BufLen
    XORDWORD = struct.unpack("<I", buf[BufLen + 1 : BufLen + 1 + 4])
[0]
    print "[ ] XorDword is %d" % XORDWORD
    XORDWORD = swap32(XORDWORD)
    print hex(XORDWORD)
    x = 0
    dst = array.array("B")

    for i in buf[1:BufLen + 1]:
        x = ord(i)

        x = x ^ (XORDWORD & 0xff)

        XORDWORD = (XORDWORD >> 8 )
        x = x ^ (XORDWORD & 0xff)

        XORDWORD = (XORDWORD >> 8 )
        x = x ^ (XORDWORD & 0xff)

        XORDWORD = (XORDWORD >> 8 )
        x = x ^ (XORDWORD & 0xff)

        x = x - (1 << 8)
        x = -x & 0xff
        print chr(x- 1),
        dst.append ((x-1))
        XORDWORD = swap32(struct.unpack("<I", buf[BufLen + 1 :
BufLen + 1 + 4])[0])
    return dst.tostring()

```

An HTTP request from Smoke Loader to the botnet controller (C&C server) consists of some internal constants as well as system information from the infected machine. The request is formatted as shown below.

```

RequestType = a2;
v6 = 63;
v13 = 63;
if ( SavedPayloadHash )
{
    v7 = strlenA(SavedPayloadHash);
    v6 = v7 + 63;
    v13 = v7 + 63;
}
AllocHeapX(v6 + 1);
v9 = (int)v8;
*v8 = 2018;
lstrcatA(v8 + 1, &MutexName_HexBuffer);
lstrcatA(v9 + 43, &unk_10005079);
*( _BYTE *) (v9 + 49) = unk_10005293 + 16 * unk_1000528F;
*( _BYTE *) (v9 + 50) = unk_1000539F;
*( _BYTE *) (v9 + 51) = ProcessTokenInfo;
*( _WORD *) (v9 + 52) = RequestType;
*( _DWORD *) (v9 + 54) = Const2;
*( _DWORD *) (v9 + 58) = InstByte;
if ( SavedPayloadHash )
    lstrcatA(v9 + 62, SavedPayloadHash);
v10 = ConnectToC2((int)&v13, (void *)1, 1);

```

### C2 packet format

The HTTP response from the botnet controller (C&C server) is typically an RC4 encrypted payload that can include multiple, so called “plugins” (such as the coin miner mentioned by the Windows Defender Team). The RC4

encrypted payload also includes one of the following commands:

- **i** - Download a file from http location field from using command ID 102
- **r** - Uninstall Dofail from system ( followed by ack packet using command ID 114)
- **u** - Update dofail ( download from http location field updated binary )

Based on way Smoke Loader calculates the mutex name an infected machine, we can create a vaccine to prevent Smoke Loader from infecting a machine:

```

#define WIN32_LEAN_AND_MEAN
#include <windows.h>

#include <wincrypt.h>

void MD5(BYTE* data, ULONG len, unsigned char *out)
{
    HCRYPTPROV hProv = 0;
    HCRYPTPROV hHash = 0;
    BYTE rgbHash[16]= {0};
    DWORD cbHash = 16;
    char hash[3] = {0};
    int i = 0;
    CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_FULL,
CRYPT_VERIFYCONTEXT);
    CryptCreateHash(hProv, CALG_MD5, 0, 0, &hHash);

    CryptHashData(hHash, data, len, 0);

    CryptGetHashParam(hHash, HP_HASHVAL, rgbHash, &cbHash, 0);
    for (i = 0 ; i < 16; i++)
    {
        sprintf(hash, "%.2X", rgbHash[i]);
        strcat(out, hash);
    }
    CryptDestroyHash(hHash);
    CryptReleaseContext(hProv, 0);

}

int main(int argc, char **argv)
{
    unsigned char *Source = (unsigned char *)
malloc(sizeof(char) * 265);
    unsigned char *md5Sum = (unsigned char *)
malloc(sizeof(char) * 34);
    DWORD lpVolumeSerialNumber = 0;

    unsigned char *FtString = (unsigned char *)
malloc(sizeof(char) * 34);

    int ComNameSize = 16;
    char CompName[MAX_COMPUTERNAME_LENGTH + 0x10] = {0};

    memset(md5Sum, 0x00, 34);
    memset(FtString, 0x00, 34);
    memset(Source, 0x00, 265);

    GetComputerName(CompName, &ComNameSize);

    GetSystemDirectoryA(Source, 260);

    Source[3] = 0x00;
    GetVolumeInformationA(Source, 0, 0, &lpVolumeSerialNumber,
0, 0, 0, 0);

    sprintf(FtString, "%s%08X%08X", CompName, 0xFEE7D621,
lpVolumeSerialNumber);

    MD5(FtString, strlen(FtString), md5Sum);
}

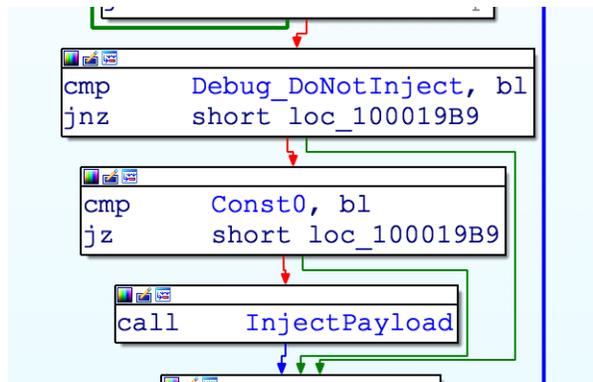
```

```

    sprintf(md5Sum, "%s%08X", md5Sum, lpVolumeSerialNumber);
    printf("%s", md5Sum);
    CreateMutex(0,0,md5Sum);
    while(1) Sleep(0x1000);
}

```

During the binary code analysis, Spamhaus Malware Labs found some sections in the code that are obviously being used by the author of Smoke Loader for debug purpose. This proves that Smoke Loader is still under heavy development of its authors and is constantly evolving.



*Debug variables*

## Conclusion

Since late 2017, Spamhaus Malware Labs could identify more than 8,000 smoke loader malware samples which call out to over 1,000 unique botnet controllers (C&C servers). In addition, to the latest code changes made by the authors of Smoke Loader in response to the countermeasures by Windows Defender, we do also see a trend in certain Smoke Loader campaigns that are shifting away from the official TLDs over to decentralized TLDs (dTLDs) such as Namecoins .bit. By using decentralized TLDs for botnet C&C hosting, botnet operators try to make their botnet C&C infrastructure more resilient against takedown attempts by security researchers and law enforcement agencies (LEA).

Spamhaus Malware Labs continues to follow the further development of Smoke Loader and takes the appropriate actions to protect Spamhaus users from this threat.

## Further reading

- [Microsoft Secure: Behavior monitoring combined with machine learning spoils a massive Dofail coin mining campaign](#)
- [Microsoft Secure: Poisoned peer-to-peer app kicked off Dofail coin miner outbreak](#)
- [Microsoft Secure: Hunting down Dofail with Windows Defender ATP](#)
- [abuse.ch: .bit - The next Generation of Bulletproof Hosting](#)
- [Spamhaus Botnet Threat Report 2017](#)

## Related Spamhaus tools & products

- [Spamhaus DROP \(Don't Route Or Peer Lists\)](#)
- [Spamhaus Response Policy Zone \(RPZ\)](#)
- [Spamhaus Zero Reputation Domain \(ZRD\)](#)
- [Spamhaus Botnet + Malware Domain List](#)
- [Spamhaus Botnet Controller List \(BCL\)](#)
- [Spamhaus/Deteque Passive DNS service](#)