Graftor - But I Never Asked for This… - Malware News - Malware Analysis, News and Indicators

Published: 2017-09-05 · Archived: 2026-04-06 00:46:25 UTC

This post is authored by Holger Unterbrink and Matthew Molyett

Overview

Free software often downloaded from large freeware distribution sites is a boon for the internet, providing users with functionality that otherwise they would not be able to use. Often users, happy that they are getting something free, fail to pay attention to the hints in the licence agreement that they are receiving additional software services bundled with the freeware they desire.
Graftor aka LoadMoney adware dropper is a potentially unwanted program often installed as part of freeware software installers. We wanted to investigate the effects this software has on a user's system. According to the analysis performed in our sandbox, Graftor and the associated affiliate files it downloads perform the following functions:

- Hijacks the user's browser and injects advertising banners
- Installs other potentially unwanted applications from partners like mail.ru
- It does not ask the user, it just silently installs these programs
- Random web page text is turned into links
- Adds Desktop and Browser Quick Launch links
- User's homepage is changed
- User's search provider is changed
- Partner adware is executed and it social engineers the user to install further software
- Checks for installed AV software
- Checks for sandbox environments
- Anti-Analysis protection
- Unnecessary API calls to overflow sandbox environments
- Creates/Modifies system certificates
- Functionality

One of the first actions of the software is to install additional software on the user's desktop, and change browser settings to point to third party websites (Fig. 1):

Looking at the Cisco Umbrella DNS data for the CnC domain used in this campaign, we can see that the campaign only lasted for a couple of days (Fig. 2a), but affected a significant number of people. Fig. 2b and 2c show domains of two of the affiliate applications which Graftor installed during our sandbox run. It is very likely that this includes users who didn't intend to install these additional applications.

Regularfood[.]gdn (Command and Control Server Domain)

Affiliates (programs installed by Graftor):

Technical Details

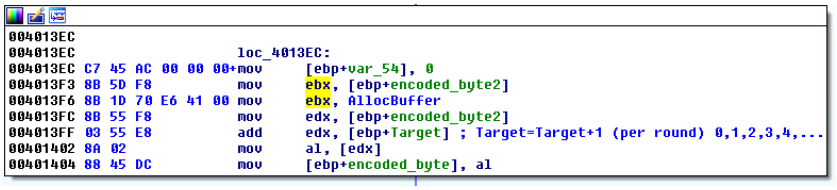
A few minutes after executing the original Graftor dropper (*2263387661.exe*), the software downloaded and installed a series of additional executables. This results in the process tree looking like this (Fig.3):

We analysed the Graftor dropper/downloader (*2263387661.exe*). It comes with multiple stages of obfuscation. The first unpacking stage of the executable uses a heavily obfuscated but fairly simple unpacking algorithm which we will describe in the following section.

This algorithm is obfuscated in the *WinMain* function distributed over several sub functions. Fig.4 shows you the complexity of the *WinMain* function in IDA, many of these building blocks are combined with further sub functions, jumping back and forth, which makes analysis particularly challenging.

First, a new buffer is allocated (see Fig.5 at 00401395) :

Then the bytes from 00416B6A (see Fig. 9 below) are decoded by different sub functions within the *WinMain* function. For example see *loc_4013EC* in Fig.6.
The code avoids calling functions by address values, but instead calls them via the values stored in registers or variables. For example the *call ebx* instruction in Fig. 5 at 00401395 results in a *VirtualAlloc* call. This makes the static analysis of the code harder. E.g without deeper analysis it is difficult to identify the destination of the *call* at 00401395 shown in Fig. 5.



2oQaIG-Zdx2J46h3lhEc9Y4J2vyoTnr04shUjUCHIABnegWcDD3iFYwDJfGZx0P6gzZ8Np_-7H4PpzHR96i0U9P1ejJ4Qpm3p01QfQbCk2NRH78qgGWnkhkc
7.21 KB

Fig. 6

Finally the decoded bytes are handed over to a function (Fig. 7 *write_unpkd_bytes2buf*), which writes these bytes into a buffer. This is the buffer which was allocated in Fig.5 at 00401395. The decoding loop starts again until all bytes are decoded:

Fig. 8 shows the *write_unpkd_bytes2buf* function itself:

The end result is that despite all of the complexity and obfuscation, the unpacking algorithm is remarkably simple and translates to the following pseudo-code (see Fig. 9 comments):

This **first stage** of unpacking extracts the code into memory. After successfully unpacking this code it is executed via *call ecx* (see Fig. 10) - the **second stage** of the unpacker:

This second stage code is position independent. It is loaded into a random address space picked by the operating system. The *VirtualAlloc* function in Fig.5 which we have mentioned above, is called with *LPVOID lpAddress* set to *NULL*, which means that the system determines where to allocate the memory region. This second stage is even more obfuscated by spaghetti code than the first stage. It's main task is to rebuild the *Import Address Table (IAT)* and resolve the addresses of certain library functions (Fig. 11), plus modify the original PE file.

It stores the function addresses in different local variables. These are passed as arguments to several setup functions, for example: change memory region 0x400000 - 0x59C000 to read/write/execute (see Fig. 12). In other words, change the whole .text, .rdata, .data, and .rsrc section of the original PE file to read/write/execute. This enables the dropper to modify and execute the code stored in these regions. As we have already seen, in order to frustrate static analysis, most calls are obfuscated by either calling registers or variables (Fig.12).

Next step at 002A14F6 is to allocate a buffer located at 01DC0000:

This buffer is filled with the bytes copied from 0042d049 from the original packed PE file:

This data is an encoded PE file. After copying the bytes to memory, it decodes them and writes them back to the buffer (Fig. 16a) at 01DC0000 (Fig. 16b)

This stage is protected with an Anti-Debugging technique. The executable uses the following two GetTickCount calls to measure the time between the two calls (Fig. 17a and 17b). If it takes too long the executable will crash.

After resolving more library function addresses and fixing the IAT of the PE file in memory, it sleeps for 258 milliseconds and jumps back to 004897D3, which we will call the **third stage** from now on.

The 2nd unpacking stage, the one we have just discussed, also decodes the URL which is later used to contact the command and control server.
First it allocates a buffer e.g. at 002B0000 (Fig. 19a) and reads the encrypted URL from the original sample at 004020c0, decodes it and stores it in the allocated buffer i.e. 002B0000 again (Fig. 19b).

The third stage (see above) is a C++ executable compiled with Visual Studio. Global object initializers allow custom classes to run during the C runtime initialization, before the apparent *WinMain* entry point. Organizing code in this way allows the malware to prepare the system survey in a way that is hidden from analysts who commence their analysis from *WinMain*.
Later, when the associated code is used, the execution is masked by memory redirection and virtual function calls.
Below you can see the callback function addresses stored in the *.rdata* segment of the PE file (Fig.20) and its initialization function *InitCallbacks* (Fig.21 and Fig. 23).

```
_rdata          segment para public 'DATA' use32
                assume cs:_rdata
                ;org 4CB5ACh
CinitFunctions  dd 0                    ; DATA XREF: __cinit+4B↑o
                dd offset sub_4CAA5D
                dd offset sub_4CAA73
                dd offset sub_4CAA89
                dd offset sub_4CAA51
                dd offset Callbacks_crt_init
                dd offset sub_4CA724
                dd offset sub_4CA73A
                dd offset sub_4CA750
                dd offset sub_4CA766
                dd offset sub_4CA77C
```

Fig. 20

```
004CA70C
004CA70C
004CA70C
004CA70C                        Callbacks_crt_init proc near
004CA70C 000 E8 3D 7E FF FF     call    CreateCallbacks
004CA711 000 8B C8              mov     ecx, eax
004CA713 000 E8 7D 75 F3 FF     call    InitCallbacks
004CA718 000 68 95 AA 4C 00     push    offset CallbacksDestroy ; void (__cdecl *)()
004CA71D 004 E8 98 E8 FB FF     call    _atexit
004CA722 004 59                 pop     ecx
004CA723 000 C3                 retn
004CA723                        Callbacks_crt_init endp
004CA723
```

Fig. 21

From the pre-*WinMain* C Run Time library (CRT) initialization, the Callback function list gets created and populated with an association of named strings (e.g. “OS”), later observed in the CnC traffic and several system information collection callback functions. For example a "systemFS" string in the CnC traffic, leads to a call to the *Graftor_CollectSystemVolumeInformation* function or "OS" triggers the call of *Graftor_CollectWindowsInformation*. Fig. 22 shows an example of such function calls and pseudo code which would lead to a similar assembler code as discussed.

```
loc_4C25B2:
or      ebx, 0FFFFFFFFh
push    offset aSystemfs ; "systemFS"
lea     esi, [esp+60h+WideString] ; std::widestring *
mov     [esp+60h+Status], ebx
call    std__widestring__set_wstring
mov     eax, esi
push    eax             ; std::widestring *
mov     ecx, edi        ; CallbackList *
mov     [esp+60h+Status], 2
call    FindCallback    ; operator[]<std::widestring>
push    0               ; int
push    1               ; free
mov     dword ptr [eax], offset Graftor_CollectSystemVolumeInformation
mov     [esp+64h+Status], ebx
call    std__widestring__reset
push    offset aOs      ; "OS"
lea     esi, [esp+60h+WideString] ; std::widestring *
call    std__widestring__set_wstring
mov     eax, esi
push    eax             ; std::widestring *
mov     ecx, edi        ; CallbackList *
mov     [esp+60h+Status], 3
call    FindCallback    ; operator[]<std::widestring>
push    0               ; int
push    1               ; free
mov     dword ptr [eax], offset Graftor_CollectWindowsInformation
```

```
std::map <std::basic_string<wchar_t>, void*> CallbackList;
```

```
CallbackList[L"systemFS"] = &Graftor_CollectSystemVolumeInformation;
CallbackList[L"OS"] = &Graftor_CollectWindowsInformation;
```

Fig. 22

The created list is linked to a global address location, which is later linked back again to local variables.

Such redirection is subtle in source code, but the resulting execution means that chains of memory accesses are seen instead of just nice clean references to the object.

Later on, a string is passed along to look up the callback and call it indirectly (Fig.25).

By using *std::basic_string<wchar_t>* instead of just plain *wchar_t* arrays, every string interaction adds two function calls and indirection. Instead of the analyst seeing a wide string being pushed to one function, it is instead a series of three. Before significant markup is performed (or when viewed in a debugger) this is just a mess of function calls and memory manipulation. Complicating the matter is that the std library is included rather than dynamically linked, so the analyst doesn't get dll calls as hints.
Further on, this 3rd stage is protected by another anti-debugging technique: the sample registers a VectoredExceptionHandler for FirstChanceExceptions (C0000005) as you can see in Fig. 26 and 27:

Then it marks the code section as PAGE_NOACCESS.

This means an exception is triggered for every single instruction in this section. The exception handler function (see Fig. 27 above) overwrites the PAGE_NOACCESS access right for the memory location which caused the exception, with a PAGE_EXECUTE_READWRITE, so it can be executed. Then the exception handler function returns to the initial instruction, it can now be executed, but the next instruction is still protected by PAGE_NOACCESS and will cause the next exception. With a debugger attached, this interrupts the debugging session for every instruction. Even if the exceptions are directly passed back to the executable, it massively slows down the execution speed.

At 004BB3FA the software starts preparing the internet request to the CnC server and encrypts the collected information to perform a GET request (Fig. 29a-c):

Talos has decrypted the GET request that is sent to the CnC server. The decoded content consists of a JSON file, which you can download here.

The executable is capable of sending the following informations to the C2 server:

MAC, SID, HD serial number, username, GUID, hostname, HD size, HD devicename, Filesystem, OS version, browser version, DotNET version, Video Driver, Language Settings, Memory, system bios version, domainname, computername, several processor related parameters, number of processors, other installed adware and unwanted programs, running processes, keyboard settings, Antispyware, Firewall, Antivirus and more.

The server responds to this with an encrypted configuration file which is processed here:

The same decryption algorithm which is used for the GET request, is also used to decrypt the CnC servers response. It generates a fairly simple stream seeded by the first byte of the packet and XORs it with the data. Underneath the encryption is a simple gzip stream.
The full decrypted file can be downloaded here. It contains the adware and other unwanted programs the Graftor downloader is supposed to install for it's partners/customers. You can see an example in Fig. 31.

The first URL from the 'l' key is used to download the partner executable and install it. The 'a' key is used as its command line parameters. We have yet to identify the exact meaning of all the keys; they are passed as parameters to a quite large JSON library. This library is also statically compiled into the binary. Besides the JSON library we also found a statically compiled SQLite library, we haven't fully investigated how it is used by the executable. However at this point we have enough information to detect and stop this adware downloader.
The information presented so far clearly shows the sophistication of this piece of software. With the data presented in the two decoded files, you have a good idea of the capabilities of the software and the impact it has on infected systems. Graftor, and the applications that it downloads also heavily check for AV products and use various techniques to detect if it is running in a sandbox environment. These are very similar to techniques commonly observed in malware.

The software makes many excessive API calls such as the following (Fig. 28) which has the effect of polluting sandbox analysis.

Conclusion

Graftor continues to be one of the most notorious potentially-unwanted-software downloaders we see in the wild. Users may be unaware that it is being bundled and executed as part of the freeware installation, since these installation files silently execute Graftor alongside the freeware.
Once Graftor is running, it exfiltrates a huge amount of user and machine identifiable information and installs additional potentially-unwanted-applications from its partners. The downloader requests administrative rights on the local machine, with this access, it can do anything it wants to do on the user's machine.
Solutions such as AMP for endpoints and AMP on network devices give administrators visibility of when software such as Graftor, and the further packages it downloads, are installed on devices. Similarly, network based detection can identify and block the CnC activity (Snort SID 44214). Thought should be given to blocking access to freeware websites to prevent the download of the Graftor installer. However, much freeware does not come bundled with Graftor and may be of great use to some users.
At the end of the day, keep in mind that if the software is free, you might be the product. Anyone using freeware should

closely review the EULA before installing it. We know it is painful, but trying to remove this kind of software is likely more painful.

Coverage

PRODUCT	PROTECTION
AMP	✔
CWS	✔
Email Security	✔
Network Security	✔
Threat Grid	✔
Umbrella	✔
WSA	✔

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors.
CWS or WSA web scanning prevents access to malicious websites and detects malware used in these attacks.
Email Security can block malicious emails sent by threat actors as part of their campaign.
Network Security appliances such as NGFW, NGIPS, and Meraki MX with Advanced Security can detect malicious activity associated with this threat.AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products. Umbrella prevents DNS resolution of the domains associated with malicious activity. Stealthwatch detects network scanning activity, network propagation, and connections to CnC infrastructures, correlating this activity to alert administrators.

IOC

Alternate Data Streams(ADS):
C:\Users\dex\AppData\Local\Temp\2263387661.exe:Zone.Identifier
C:\Users\dex\AppData\Local\Temp\QBPO5ppcuhJG.exe:tmp
C:\Users\dex\AppData\Local\Temp\2263387661.exe:tmp
C:\Users\dex\AppData\Local\Temp\AyWdp7tHPIeU.exe:tmp
C:\Windows\System32\regsvr32.exe:Zone.Identifier

Hashes:

2263387661.exe (Graftor Dropper)
9b9ce661a764d84a4636812e1dfcb03b (MD5)
Fd3ccf65eab21a77d2e440bd23c59d52e96a03a4 (SHA1)
41474cd23ff0a861625ec1304f882891826829ed26ed1662aae2e7ebbe3605f2 (SHA256)

Dumped 2nd stage:
40bde09fc059f205f67b181c34de666b (MD5)
99c7627708c4ab1fca3222738c573e7376ab4070 (SHA1)
Eefdbe891e35390b84181eabe0ace6e202f5b2a050e800fb8e82327d5e57336d (SHA256)

Dumped 3rd stage:
1e9f40e70ed3ab0ca9a52c216f807eff (MD5)
7c4cd0ff0e004a62c9ab7f8bd991094226eca842 (SHA1)
5eb2333956bebb81da365a26e56fea874797fa003107f95cda21273045d98385 (SHA256)

URLs:

Command and Control Server GET Request:
hxxp://kskmasdqsjuzom[.]regularfood[.]gdn/J/ZGF0YV9maWxlcz0yMyZ0eXBlPXN0YXRpYyZuYW1lPVRlbXAlNUMyMjYzMzg3NjYxLmV4ZSZyt

Set-Cookie: GSID=3746aecf3b94384b9de720158c4e7d88; expires=Sat, 12-Aug-2017 15

Command and Control Server POST Request
hxxp://kskmasdqsjuzom[.]regularfood[.]gdn/J/ZGF0YV9maWxlcz0yMyZ0eXBlPXN0YXRpYyZuYW1lPVRlbXAlNUMyMjYzMzg3NjYxLmV4ZSZyt

Set-Cookie: GSID=3746aecf3b94384b9de720158c4e7d88; expires=Sat, 12-Aug-2017 15

Domains from sandbox run:
arolina[.]torchpound[.]gdn
binupdate[.]mail[.]ru
crl[.]microsoft[.]com
dreple[.]com

gambling577[.]xyz
jvusdtufhlreari[.]twiceprint[.]gdn
kskmasdqsjuzom[.]regularfood[.]gdn
mentalaware[.]gdn
mrds[.]mail[.]ru
nottotrack[.]com
plugpackdownload[.]net
s2[.]symcb[.]com
sputnikmailru[.]cdnmail[.]ru
ss[.]symcd[.]com
xml[.]binupdate[.]mail[.]ru

Snort Rules:
SID 44214
