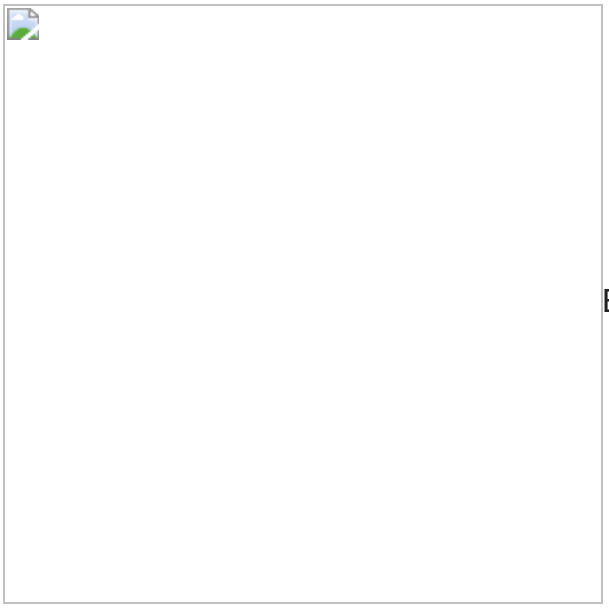
Analysis: Ursnif - spying on your data since 2007

blog.gdatasoftware.com/2016/11/29325-analysis-ursnif-spying-on-your-data-since-2007

A game of cat and mouse has been going on ever since the first ever malware started circulating in the wild and the first Antivirus appeared on the market. Although it may seem that brand new malware families appear on a daily basis, the truth looks somewhat different. A lot of the malware which is in circulation is a reiteration of something that has existed for quite some time. After all, malware development costs criminals time and money, and if they can lower the cost by reusing something preexisting, they will do so. In this analysis we will take a look as just such a case.

Infection vector

As in other previous cases, Ursnif can only be installed after initiating user interaction. The user is asked to open an email attachment which looks like an Office document. The document claims that it can only be displayed properly if macros are enabled by the users. Should the user comply, a malicious VB script will be dropped in the %user temp% folder.



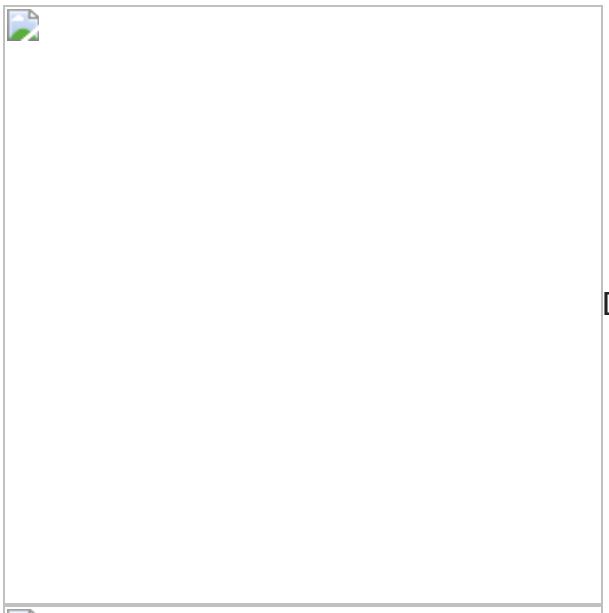
Email with a malicious attachment containing Ursnif

Looking at the script: Analysis prevention

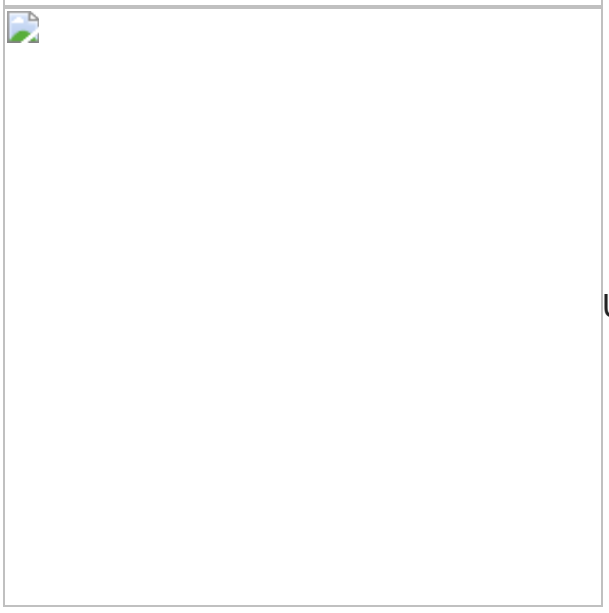
When running the macro, an encoded VB script is dropped. It is decoded automatically on execution and downloads the actual executable payload. The script contains obfuscated VB code with some garbage data added to confuse analysts. The executable payload is encrypted (1-byte XOR). The decryption is also done by the VB script. To further throw analysts off its track, each sample of Ursnif is packed differently. In addition to this, the latest versions also have a mechanism which determines whether the executable is running in a

virtual machine. To check if this is the case, several system parameters are queried. In this case, the hardware information of the disk drive is queried using SetupDiGetClassDevsA to open a handle to device information set as well as two other APIs (SetupDiEnumDeviceInfo and SetupDiGetDeviceRegistryProperyA) .

If Ursnif finds itself running inside a virtual machine, it will just terminate and take no further action. The purpose of this is to hinder analysis - most analysts tend to use virtual machines for their work.



Decoded script from the email attachment



Ursnif probing for hardware information to determine if it is running in a VM.

To top

C2 communications

Like many types of malware, this one also communicates with a Command & Control (C2) server. In order to make it more difficult to pinpoint those, the domain names are often generated locally as opposed to hard-coding them. Those domain names are calculated using a Domain Generation Algorithm (DGA). What is remarkable about this malware exhibit is the way in which the domains are generated. As a 'data source' for its DGA it uses some well-known websites and takes bits of text that are used to generate C2 domains.

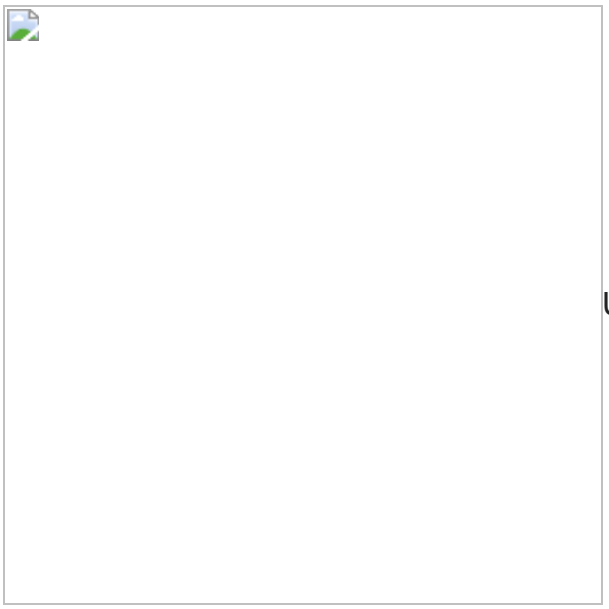
Fun fact: the authors appear to have a sense of humor, too, albeit a twisted one; among the texts used as data sources, there is one from Apple's website (opensource.apple.com/source/Security/Security-29/SecureTransport/LICENSE.txt) as well as RFC 1237 (Guidelines for OSI NSAP (Network Service Access Provider) Allocation in the Internet) from the IETF website.

Data exfiltration via HTTP POST

Ursnif is capable of collecting a user’s system activity, including keystrokes, new connected devices, running application, network traffic and browser activity. This data is written to a compressed file (ZIP compression) and stored in the %user temp% folder. It is named as {random 4 hexadecimal char}.bin.

The C2 URL contains an "/images" section, followed by a blob of encoded data as well as an extension. The encoded data blob contains additional information such as bot ID (unique identifier of the infected machine), the build version of the malware, the infected machine's OS version, its public IP, the name of the file about to be uploaded the server ID and the file type. This information is encrypted using AES 256 with an additional layer of base64 encoding. The locally stored and compressed data is then uploaded to the C2 server using HTTP POST.

To raise less suspicion, slashes are added at random; special characters are substituted with their equivalent hex value with an added underscore as a prefix. The URL then looks more or less legitimate, in case the C2 communication shows up in a log file.



Ursnif communicating with its C2 server. The long URL with the encoded and encrypted data is visible at the top.

Persistence, Hooking & Processing of External Commands

Once installed, Ursnif attempts to inject its components into a running instance of explorer.exe. Should this fail, it will start a new instance of svchost.exe and injects its component into this process instead. Subsequently, it hooks the APIs of widely used browsers, in this case Chrome, Opera, Internet Explorer and Firefox. As soon as a user visits a predetermined banking or payment website, Ursnif performs a web inject to steal any login credentials. Monitored websites include (but are not limited to): *wellsfargo.com*, *.bankofamerica.com*, *paypal.com*, www.amazon.com/ap/signin*, *.americanexpress.com*, www.chase.com*, www.discover.com and online.cit*.com*

Ursnif is also known to be able to accept and execute external commands it receives from its server. Should Ursnif receive the command to perform the function LOAD_UPDATE, it will convert the command to CRC hash "0xA172B308". This value is then compared against a local list of hashed commands and will only be executed (using named pipes) if the hash value matches the local list. This is to ensure that analysts cannot just blindly feed a list of frequently used commands into a given payload to see which input triggers any reaction. The infected machine in return will report the request status as well as the execution time back to the C2 server.

Conclusion

As outlined earlier, criminals usually do not make attempts to reinvent the wheel. They stick with what has been tried and tested before rather than develop something new from the ground up. There are other examples for this as well, such as ransomware which is based on the "Hidden Tear" source code that was originally intended to be used for educational purposes.

Ursnif has been around for quite a while now. The first version appeared in 2007. Back then it was a competitor of the ZEuS banking malware. This goes to show that, at least in the world of malware, "old" does not mean "dead" or "inactive", because Ursnif is quite the opposite. Its source code is available online. This makes it easier for online criminals to add new features that suit their own needs. The reliance on well-known tools is pretty common and even applies to modern APT scenarios.

G DATA customers are protected from this on multiple levels. Besides having signatures, our BankGuard technology reliably detects any attempts by malware to hook into a browser to steal data. In addition to this, our Behavior Monitoring detects behavioral patterns which are associated with malicious software.

IoC list

IOC

Campaign with password-protected word document:

Word document file -
62443895379ab934ad0621c8a2e084414862cd6daa4fa1d09370a6d3e5de3d9b

Dropped VBS -
97d382d6eb5f2113dcbad702b43c648a34c9f2b516da27b0ce2cb2493e93171b

Payload Gozi/URSNIF -
9db26083ffe1e1c83f47464a047e46e579787bea2ae945fb865f5cc588b86229

Unpacked sample -
51f81493dd1c34c8909d65060b7e96e301e3ec38741660a1248fdc1203b543e8

32 bit DLL module -
668ac0ef90e0db8dd33020335f43505d1afce803b7e659d51e3be2bdcc933c5f

64 bit DLL module –
95921f6bde5e4d71fd4308822db46ce46b646d863037f89f236da7f0337c57d4

Other campaign:

Word document file -
371a81358595c639eb290516d763e2c9cfa1dda1506b0deb37c46504d31a79ea

Dropped VBS -
4d345043c03c09212abef68d84a82102d8409157b00bf89d9dc6f4b98238927e

Payload Gozi/URSNIF -
172f359baa478d80a9a8eccde0393e3fb8a58f0444a1b71d99d87c6a50855297

Unpacked Sample -
17af05823ed53b5e794c3a5696326454d8f91ad6af4f33e2ffa4b780bfd17d98

32 bit DLL module –
7d7f03e772c0c2117923d82b4fff5bc20d03a5fff04d0605e1a35a04dd8be34c

64 bit DLL module –
cfadd9b071c80f75eb1eedcce8e697a9ac31334abc919e286336acc01c3db089

Other campaign:

1st Stage JS Downloader –
9a44ff53471012328a3b167c149ed71c2e82b117de8f9463f5773b5b4f5cc7b6

2nd Stage Downloader –
827c1ce97229a99c9badfa79b144690c91314603f250696b80765ba8d9ad1423

3rd Stage Downloader –
d8bf69b386e80f3e4cc8c4d612900a38ca5b0f661be89cfcb3a0fe4999a96bf4

Payload Gozi/URSNIF –
4f3926e686bfda88b28cd009d1a84396fc6e0bdc070a962f91da43fbde2a29c7

Unpacked sample –
672c3cd4eb8a5e4dc974c8f14a23f23f5863c5fb8cc1043c49771de715259dde

32 bit DLL module –
92e2cefcd7c334cad5b7eac0c2c79392a4e959317cdc993420395919e19d71c5

64 bit DLL module –
579f53c1eda4fd18fb5265f399c67238da740059ae300374ee234866cc92f32f