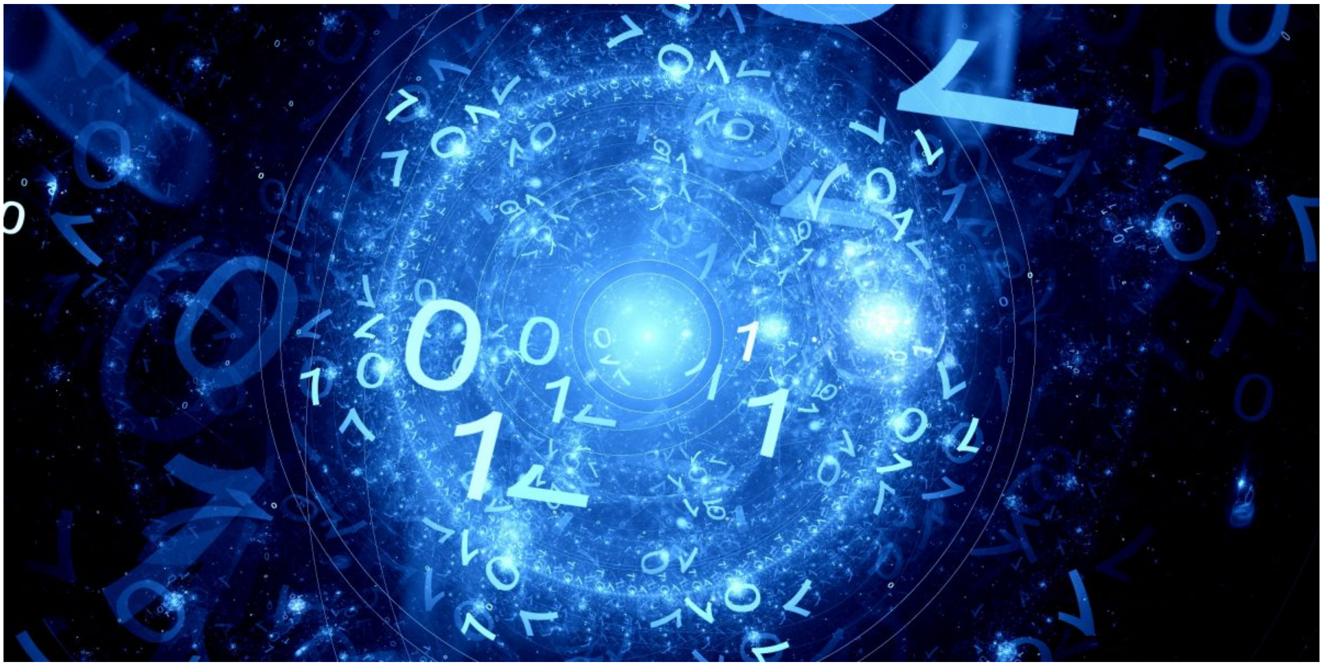
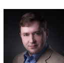
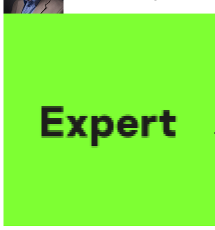
Use of DNS Tunneling for C&C Communications

securelist.com/use-of-dns-tunneling-for-cc-communications/78203/



Authors

-  Alexey Shulmin
- 

 Sergey Yunakovsky

– Say my name.

– 127.0.0.1!

– You are goddamn right.

Network communication is a key function for any malicious program. Yes, there are exceptions, such as cryptors and ransomware Trojans that can do their job just fine without using the Internet. However, they also require their victims to establish contact with the threat actor so they can send the ransom and recover their encrypted data. If we omit these two and have a look at the types of malware that have no communication with a C&C and/or threat actor, all that remains are a few outdated or extinct families of malware (such as Trojan-ArcBomb), or irrelevant, crudely made prankware that usually does nothing more than scare the user with screamers or switches mouse buttons.

Malware has come a long way since the Morris worm, and the authors never stop looking for new ways to maintain communication with their creations. Some create complex, multi-tier authentication and management protocols that can take weeks or even months for analysists to decipher. Others go back to the basics and use IRC servers as a management host – as we saw in the recent case of Mirai and its numerous clones.

Often, virus writers don't even bother to run encryption or mask their communications: instructions and related information is sent in plain text, which comes in handy for a researcher analyzing the bot. This approach is typical of incompetent cybercriminals or even experienced programmers who don't have much experience developing malware.

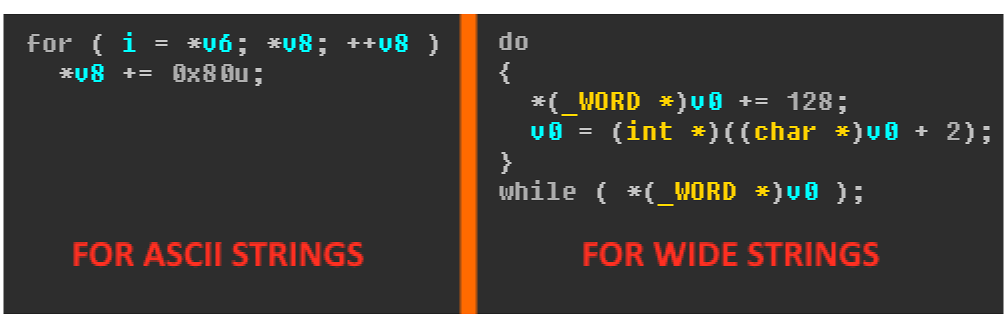
However, you do get the occasional off-the-wall approaches that don't fall into either of the above categories. Take, for instance, the case of a Trojan that Kaspersky Lab researchers discovered in mid-March and which establishes a DNS tunnel for communication with the C&C server.

The malicious program in question is detected by Kaspersky Lab products as Backdoor.Win32.Denis. This Trojan enables an intruder to manipulate the file system, run arbitrary commands and run loadable modules.

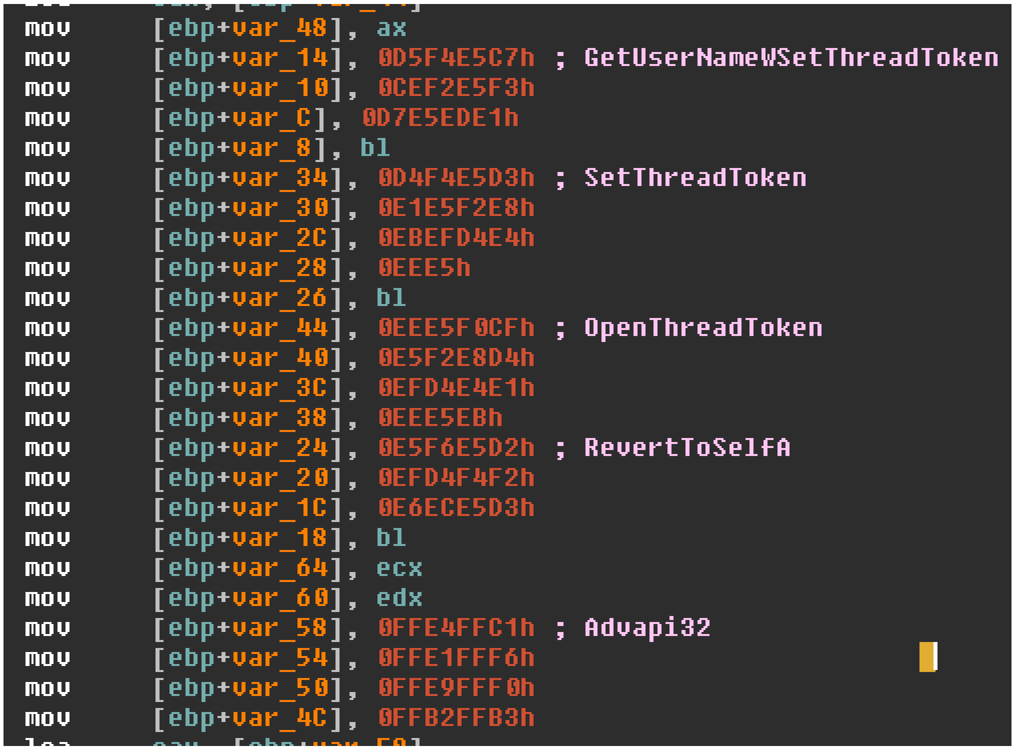
Encryption

Just like lots of other Trojans before it, Backdoor.Win32.Denis extracts the addresses of the functions it needs to operate from loaded DLLs. However, instead of calculating the checksums of the names in the export table (which is what normally happens), this Trojan simply compares the names of the API calls against a list. The list of API names is encrypted by subtracting 128 from each symbol of the function name.

It should be noted that the bot uses two versions of encryption: for API call names and the strings required for it to operate, it does the subtraction from every byte; for DLLs, it subtracts from every other byte. To load DLLs using their names, LoadLibraryW is used, meaning wide strings are required.



'Decrypting' strings in the Trojan



```
mov     [ebp+var_48], ax
mov     [ebp+var_14], 0D5F4E5C7h ; GetUserNameWSetThreadToken
mov     [ebp+var_10], 0CEF2E5F3h
mov     [ebp+var_C], 0D7E5EDE1h
mov     [ebp+var_8], bl
mov     [ebp+var_34], 0D4F4E5D3h ; SetThreadToken
mov     [ebp+var_30], 0E1E5F2E8h
mov     [ebp+var_2C], 0EBEFD4E4h
mov     [ebp+var_28], 0EEE5h
mov     [ebp+var_26], bl
mov     [ebp+var_44], 0EEE5F0CFh ; OpenThreadToken
mov     [ebp+var_40], 0E5F2E8D4h
mov     [ebp+var_3C], 0EFD4E4E1h
mov     [ebp+var_38], 0EEE5EBh
mov     [ebp+var_24], 0E5F6E5D2h ; RevertToSelfA
mov     [ebp+var_20], 0EFD4F4F2h
mov     [ebp+var_1C], 0E6ECE5D3h
mov     [ebp+var_18], bl
mov     [ebp+var_64], ecx
mov     [ebp+var_60], edx
mov     [ebp+var_58], 0FFE4FFC1h ; Advapi32
mov     [ebp+var_54], 0FFE1FFF6h
mov     [ebp+var_50], 0FFE9FFF0h
mov     [ebp+var_4C], 0FFB2FFB3h
```

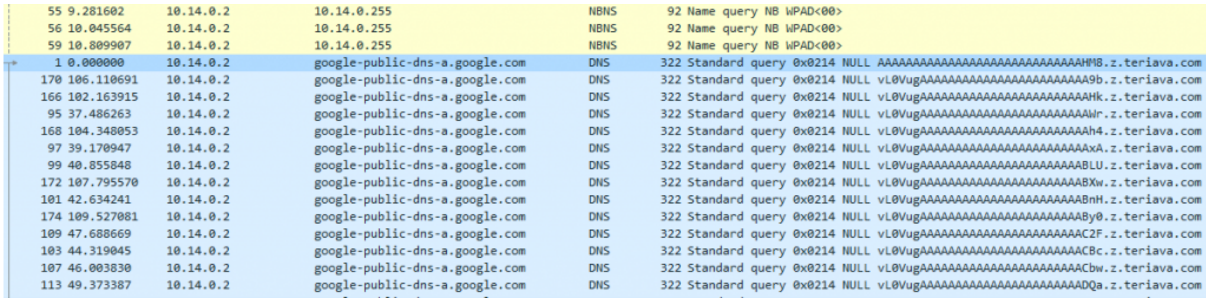
Names of API functions and libraries in encrypted format

It should also be noted that only some of the functions are decrypted like this. In the body of the Trojan, references to extracted functions alternate with references to functions received from the loader.

C&C Communication

The principle behind a DNS tunnel's operation can be summed up as: "If you don't know, ask somebody else". When a DNS server receives a DNS request with an address to be resolved, the server starts looking for it in its database. If the record isn't found, the server sends a request to the domain stated in the database.

Let's see how this works when a request arrives with the URL Y3VyaW9zaXR5.example.com to be resolved. The DNS server receives this request and first attempts to find the domain extension '.com', then 'example.com', but then it fails to find 'Y3VyaW9zaXR5.example.com' in its database. It then forwards the request to example.com and asks it if such a name is known to it. In response, example.com is expected to return the appropriate IP; however, it can return an arbitrary string, including C&C instructions.

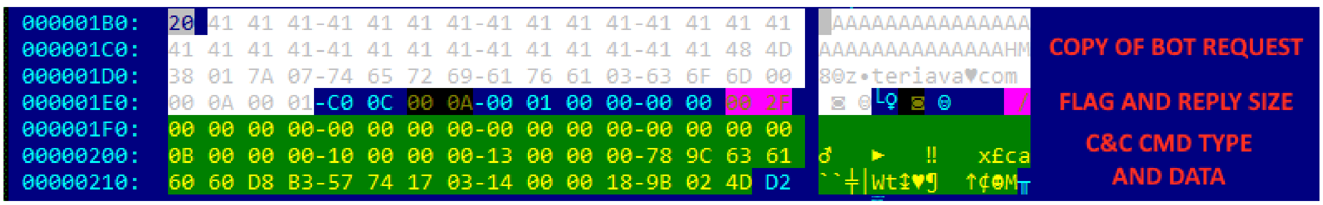


55	9.281602	10.14.0.2	10.14.0.255	NBNS	92 Name query NB WPAD<00>
56	10.045564	10.14.0.2	10.14.0.255	NBNS	92 Name query NB WPAD<00>
59	10.809907	10.14.0.2	10.14.0.255	NBNS	92 Name query NB WPAD<00>
1	0.000000	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAHM8.z.teriava.com
170	106.110691	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL vL0VugAAAAAAAAAAAAAAAAAAAAAAAAA9b.z.teriava.com
166	102.163915	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL vL0VugAAAAAAAAAAAAAAAAAAAAAAAAAHk.z.teriava.com
95	37.486263	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL vL0VugAAAAAAAAAAAAAAAAAAAAAAAAAWr.z.teriava.com
168	104.348053	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL vL0VugAAAAAAAAAAAAAAAAAAAAAAAAAh4.z.teriava.com
97	39.170947	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL vL0VugAAAAAAAAAAAAAAAAAAAAAAAAAxA.z.teriava.com
99	40.855848	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL vL0VugAAAAAAAAAAAAAAAAAAAAAAAABLU.z.teriava.com
172	107.795570	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL vL0VugAAAAAAAAAAAAAAAAAAAAAAAABXw.z.teriava.com
101	42.634241	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL vL0VugAAAAAAAAAAAAAAAAAAAAAAAABnH.z.teriava.com
174	109.527081	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL vL0VugAAAAAAAAAAAAAAAAAAAAAAAABy0.z.teriava.com
109	47.688669	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL vL0VugAAAAAAAAAAAAAAAAAAAAAAAAC2F.z.teriava.com
103	44.319045	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL vL0VugAAAAAAAAAAAAAAAAAAAAAAAACBc.z.teriava.com
107	46.003830	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL vL0VugAAAAAAAAAAAAAAAAAAAAAAAACbw.z.teriava.com
113	49.373387	10.14.0.2	google-public-dns-a.google.com	DNS	322 Standard query 0x0214 NULL vL0VugAAAAAAAAAAAAAAAAAAAAAAAADQa.z.teriava.com

Dump of Backdoor.Win32.Denis traffic

This is what Backdoor.Win32.Denis does. The DNS request is sent first to 8.8.8.8, then forwarded to z.teriava[.]com. Everything that comes before this address is the text of the request sent to the C&C.

Here is the response:



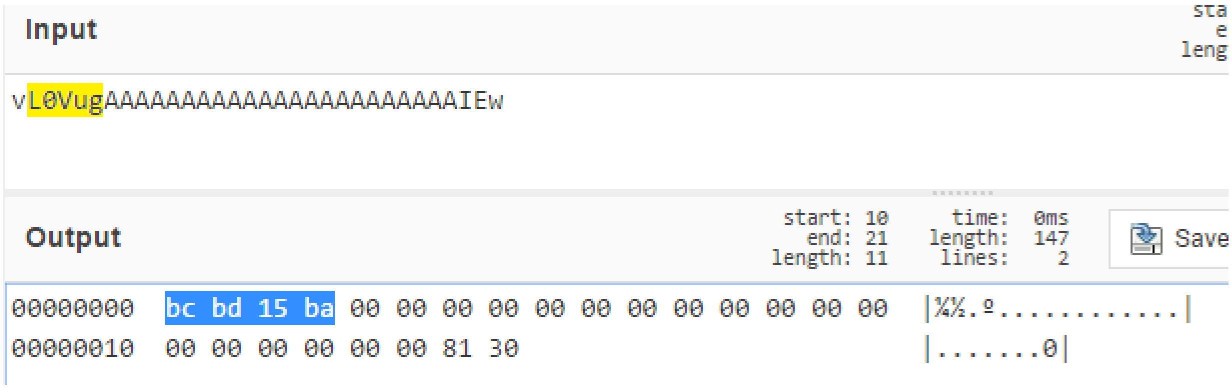
DNS packet received in response to the first request

Obviously, the request sent to the C&C is encoded with Base64. The original request is a sequence of zeros and the result of GetTickCount at the end. The bot subsequently receives its unique ID and uses it for identification at the start of the packet.

The instruction number is sent in the fifth DWORD, if we count from the start of the section highlighted green in the diagram above. Next comes the size of the data received from C&C. The data, packed using zlib, begins immediately after that.

The unpacked C&C response

The first four bytes are the data size. All that comes next is the data, which may vary depending on the type of instruction. In this case, it's the unique ID of the bot, as mentioned earlier. We should point out that the data in the packet is in big-endian format.



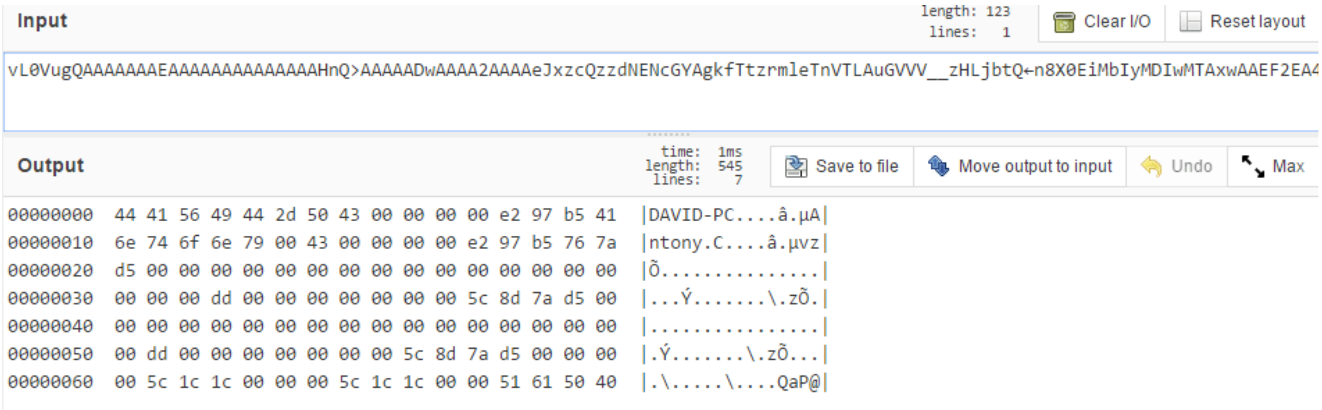
The bot ID (highlighted) is stated at the beginning of each request sent to the C&C

C&C Instructions

Altogether, there are 16 instructions the Trojan can handle, although the number of the last instruction is 20. Most of the instructions concern interaction with the file system of the attacked computer. Also, there are capabilities to gain info about open windows, call an arbitrary API or obtain brief info about the system. Let us look into the last of these in more detail, as this instruction is executed first.

```
; enum CMDS, mappedto_64
CMD_API_RUN       = 1
CMD_FREE_LIB      = 2
CMD_PROC_START    = 3
CMD_READ_FILE     = 4
CMD_SHELL_RES     = 5
CMD_NONE          = 6
CMD_WRITE         = 7
CMD_ENUM_WINDOWS  = 0Ah
CMD_SET_REG       = 0Bh
CMD_REG           = 0Ch
CMD_FIND          = 0Fh
CMDS_MOVE         = 10h
CMD_DELETE        = 11h
CMD_DRVS_INF      = 12h
CMD_CREATE_DIR    = 13h
CMD_REMOVE        = 14h
```

Complete list of C&C instructions



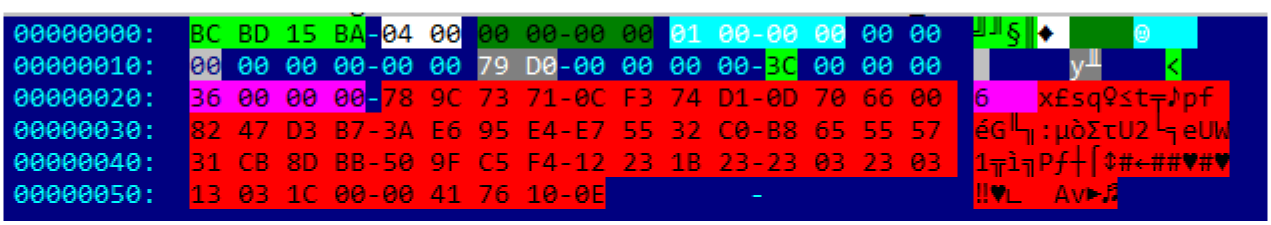
Information about the infected computer, sent to the C&C

As can be seen in the screenshot above, the bot sends the computer name and the user name to the C&C, as well as the info stored in the registry branch Software\INSUFFICIENT\INSUFFICIENT.INI:

- Time when that specific instruction was last executed. (If executed for the first time, ‘GetSystemTimeAsFileTime’ is returned, and the variable BounceTime is set, in which the result is written);
- UsageCount from the same registry branch.

Information about the operating system and the environment is also sent. This info is obtained with the help of NetWkstaGetInfo.

The data is packed using zlib.



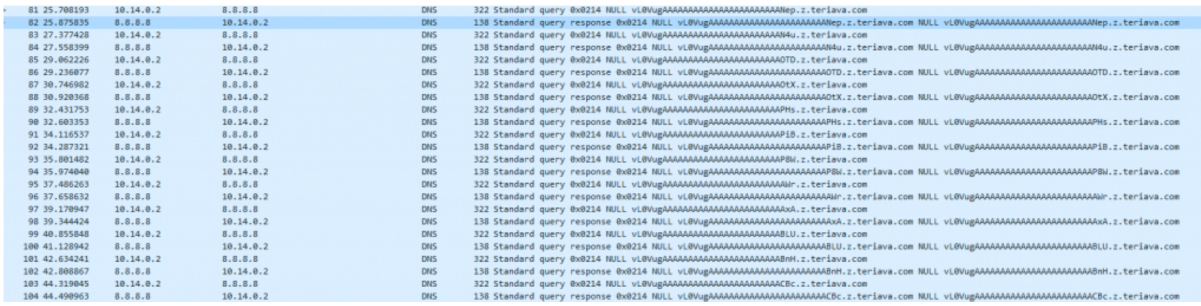
The DNS response prior to Base64 encoding

The fields in the response are as follows (only the section highlighted in red with data and size varies depending on the instruction):

- Bot ID;
- Size of the previous C&C response;
- The third DWORD in the C&C response;
- Always equals 1 for a response;
- GetTickCount();
- Size of data after the specified field;
- Size of response;

- Actual response.

After the registration stage is complete, the Trojan begins to query the C&C in an infinite loop. When no instructions are sent, the communication looks like a series of empty queries and responses.



Sequence of empty queries sent to the C&C

Conclusion

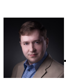
The use of a DNS tunneling for communication, as used by Backdoor.Win32.Denis, is a very rare occurrence, albeit not unique. A similar technique was previously used in some POS Trojans and in some APTs (e.g. Backdoor.Win32.Gulpix in the PlugX family). However, this use of the DNS protocol is new on PCs. We presume this method is likely to become increasingly popular with malware writers. We'll keep an eye on how this method is implemented in malicious programs in future.

MD5

facec411b6d6aa23ff80d1366633ea7a
018433e8e815d9d2065e57b759202edc
1a4d58e281103fea2a4ccbfab93f74d2
5394b09cf2a0b3d1caaecc46c0e502e3
5421781c2c05e64ef20be54e2ee32e37

- Backdoor
- DNS
- Malware Descriptions
- Malware Technologies
- Trojan

Authors

-  Alexey Shulmin

- Expert Sergey Yunakovsky

Use of DNS Tunneling for C&C Communications

Your email address will not be published. Required fields are marked *