

DRIDEX IN THE WILD

Meng Su

Tencent, China

Dridex is a descendent of the Cridex malware. Its initial spread occurred in late 2014 via spam and the malware is still active in the wild today. Dridex is a *Windows* executable which uploads system information to its C&C server before downloading a DLL. After the DLL has been installed by the executable, the C&C server will control the infected PC, sending it commands to carry out further harmful instructions. In this article, we will analyse the main executable, focusing on the following actions: obtaining APIs, getting server data, getting and encoding system information, and communicating with the C&C server.

OBTAINING APIS

All of the *Windows* APIs the bot uses are obtained by a function. The argument passed into this function is only an index. This 'index' is an index number of the API_Address array – the API-name-encode-data-block uses the same index value.

At first, the malware checks the API_Address array, which is initiated with a NULL value. If API_Address[API_index] is found with a valid value, the function returns the address. Otherwise the malware moves onto the next step.

In the second step, the malware decodes the API_Name from the API-name-encode-data-block with the API_index using an algorithm which is predefined by the malware itself. The decoded data contains two parts, DLL_index and API_Name:

```
API_Data
{
  BYTE DLL_index;
  BYTE[] API_Name;
};
```

The role of DLL_index is the same as that of the API_index. The malware has a DLL_Module array which is similar to the API_Address array and also a similar DLL-name-encode-data-block.

The malware checks the DLL_Module array. If it finds valid data at DLL_Module[DLL_index], then it returns the DLL module for the next step. Otherwise the malware will get the DLL module using the following method: similar to API_Name, the DLL_Name is decoded from the DLL-name-

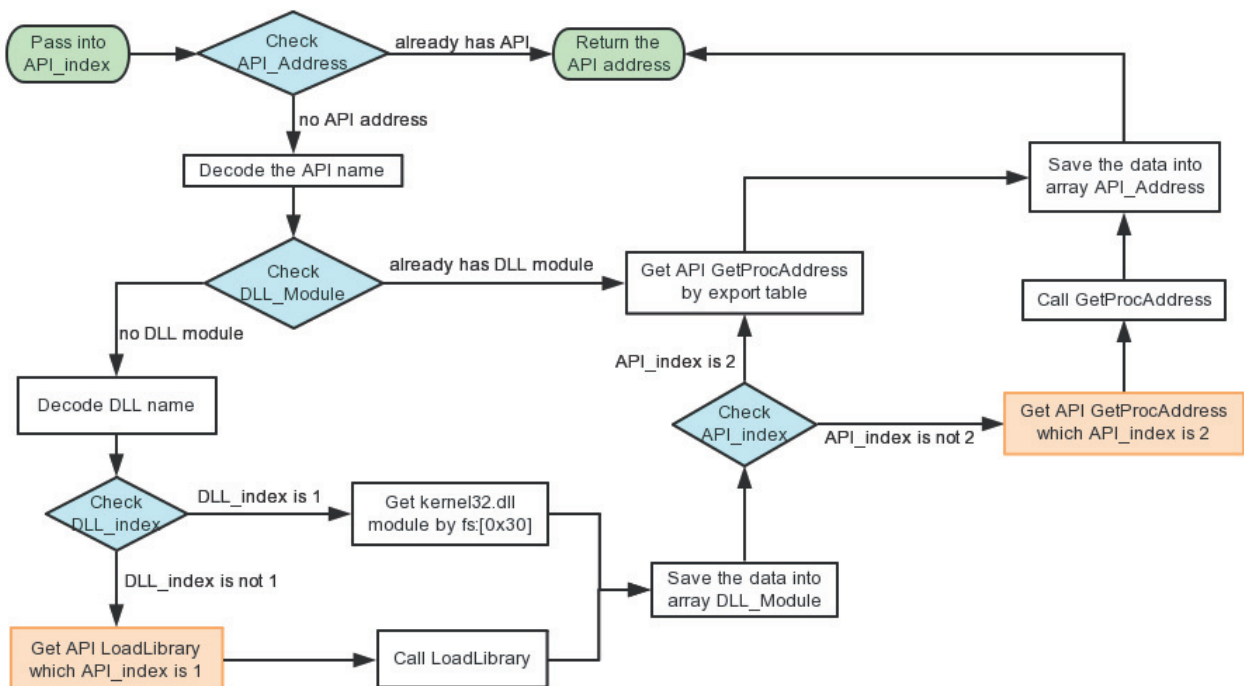


Figure 1: Obtaining APIs.

encode-data-block by DLL_index. After that, the malware checks whether the DLL_index value is equal to one. By the design of the malware, the DLL_index of kernel32.dll is one. The way to get this DLL's module is using register fs:[0x30], which points to the PEB structure, and then finding the PEB_LDR_DATA structure via the PEB. In the PEB_LDR_DATA structure we can find out the DLL base address by comparing the DLL name. If the DLL_index value is not one, the malware will get the LoadLibrary API whose API_index value is one. The malware then uses this API to get the DLL module. The malware records the DLL module into the DLL_Module array, regardless of whether or not the DLL_index is 1.

If the API_index passed is 2, which represents the GetProcAddress API in this malware, the bot will traverse the DLL's export table to get the API address. Otherwise the malware will get the GetProcAddress API first and then call this API to get the other API's address. The API address will be saved into the API_Address array.

Note that the addresses of the LoadLibrary and GetProcAddress APIs are always the first two addresses obtained by the malware. The flowchart in Figure 1 shows the full logic steps of getting any single API address.

GETTING SERVER DATA

In this malware, the C&C server address is not stored as plain text. The malware uses the GetModuleHandleW API to locate the IMAGE_DOS_HEADER, and then locates the section header. It will find one section's virtual address whose section name is '.sdata' (Figure 2).

This section contains only 0x7A valid bytes. The first DWORD (0xA9E97561 in this case) is a key which is used to XOR the other 0x76 bytes. Figure 3 shows the content after it has been XOR'ed.

As shown in Figure 3, this data is still encrypted. The 0x76 byte-long decoded data consists of three parts: the size of the

61 75 E9 A9	0F 75 E9 A9	F8 75 E9 A9	F4 94 69 AB	au	棋	u	棋	鵠	棋	會	i
5D 16 86 C7	07 1C F4 CE	41 17 37 DD	7D 10 2A 94	J	品	■	■	■	■	■	■
43 47 D9 26	21 4B E4 A3	41 F5 D5 DA	79 10 9B DF	CG?	!K	列	A	踰	趁	■	■
67 2A 85 D0	08 15 9D 8C	50 4C DD 87	58 47 D1 5B	g*	咬	■	■	■	■	■	■
17 42 1C 75	54 4F 10 32	63 46 C0 5E	CE 8B DF 95	■	■	■	■	■	■	■	■
AF 42 D9 76	96 53 21 C7	AC 47 A9 5E	1E 53 37 FD	疊	賤	■	■	■	■	■	■
1D 02 38 81	46 AA 96 C6	43 CE 91 0E	3B 60 21 CB	■	■	■	■	■	■	■	■
74 5A 8A 0F	41 F9 64 5E	E1 75 00 00	00 00 00 00	tZ?	A	鵠	■	■	■	■	■

Figure 2: .sdata section content.

6E 00 00 00	99 00 00 00	95 E1 80 02	3C 63 6F 6E	n...	?.	■	■	■	■	■	■
66 69 1D 67	20 62 DE 74	1C 65 C3 3D	22 32 30 8F	fi	g	■	■	■	■	■	■
40 3E 0D 0A	20 80 3C 73	18 65 72 76	06 5F 6C 79	@)	..	■	■	■	■	■	■
69 60 74 25	31 39 34 2E	39 32 38 F2	76 37 F5 DC	i`	t%	194.928	■	■	■	■	■
35 3A F9 9B	02 33 29 F7	AF CE 36 3C	CE 37 30 DF	5:	■	■	■	■	■	■	■
F7 26 C8 6E	CD 32 40 F7	7F 26 DE 54	7C 77 D1 28	?	高	?	?	?	?	?	?
27 DF 7F 6F	22 BB 78 A7	5A 15 C8 62	15 2F 63 A6	'?	o	■	■	■	■	■	■
20 8C 8D F7	80			實	■	■	■	■	■	■	■

Figure 3: XOR content.

95 00 00 00	3C 63 6F 6E	66 69 67 20	62 6F 74 6E	?..	<	config	botn
65 74 3D 22	32 30 30 22	3E 0D 0A 20	20 20 3C 73	et="	200"	>	..
65 72 76 65	72 5F 6C 69	73 74 3E 0D	0A 31 39 34	erver_	list	>	..
2E 32 38 2E	38 37 2E 31	32 35 3A 34	34 34 33 0D	.28.87.125:4443.			
0A 31 38 35	2E 36 36 2E	37 30 2E 34	35 3A 38 34	.185.66.70.45:84			
34 33 0D 0A	38 32 2E 31	34 36 2E 35	38 2E 32 31	43..82.146.58.21			
36 3A 38 34	34 33 0D 0A	31 38 35 2E	31 31 2E 32	6:8443..185.11.2			
34 37 2E 32	32 36 3A 38	34 34 33 0D	0A 20 20 20	47.226:8443..			
3C 2F 73 65	72 76 65 72	5F 6C 69 73	74 3E 0D 0A	</server_list>	..		
3C 2F 63 6F	6E 66 69 67	3E 00 00 00	00 00 00 00	</config>		

Figure 4: The raw data.

encoded data (0x6E), the size of the raw data (0x99), and the encoded data:

```
Encoded_Server_Data
{
  DWORD szEncodeData;
  DWORD szRawData;
  BYTE[] EncodedData;
};
```

The 'EncodedData' is compressed by the aPLib algorithm [1]. The decompressed raw data is shown in Figure 4.

The first four bytes of this raw data indicate the length of the data behind it. Figure 5 shows the configuration of the server. The 'botnet' attribute shows the botnet_id; the 'server_list' tag shows the server URLs.

```
<config botnet="200">
  <server_list>
194.28.87.125:4443
185.66.70.45:8443
82.146.58.216:8443
185.11.247.226:8443
  </server_list>
</config>
```

Figure 5: Server data.

After getting the server URL, the malware will collect system information for further communication.

GETTING AND ENCODING SYSTEM INFORMATION

The collected information will be stored in XML format in two parts. The first part is composed as follows:

```
<loader><get_module unique="%s" botnet="%d"
system="%dv" name="bot" bit="%d"/>
```

Meanwhile, the format of the other part is:

```
<soft><![CDATA[%s]]></soft></loader>.
```

In the first part, the value of the 'unique' attribute records a string relating to three registry entries:

```
Key: HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/
Control/ComputerName/ComputerName
Name: ComputerName

Key: HKEY_LOCAL_MACHINE/Volatile Environment
Name: USERNAME
```

45	34	32	33	30	37	32	30	3F	46	35	45	34	32	32	5F	E4230720?	F5E422_
34	33	30	65	63	36	61	32	61	66	33	61	64	39	39	33	430ec6a2af3ad993	
39	63	63	33	30	30	39	63	64	36	31	39	65	33	33	38	9cc3009cd619e338	

Figure 6: The content of the 'unique' argument.

```
Key: HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows
NT/CurrentVersion
Name: InstallDate
```

The malware retrieves the values of these three keys and combines them as a data block, System_Info, then calculates the MD5 of this data block. The malware also checks every character of the 'ComputerName' value. If a character is not found on a list which contains the Latin letters and some special symbols, it will be replaced with the character '?'. I think the malware author made a mistake here: there is no letter 'D' on the letter list and there is an extra 'S' – I guess that's because 'D' is pretty close to 'S' on the keyboard and this was probably a typo. This means that the malware will replace 'D' with '?'. In the end, the changed 'ComputerName' value and the MD5 of the System_Info are joined with the character '_' (Figure 6), then set with the 'unique' attribute.

The value of the 'botnet' argument is a botnet_id which is the same botnet_id as in the server configuration (Figure 5). The value of the 'system' attribute is a hash value which indicates the version of the operating system (e.g. XP or Win7), whether it is an NT kernel or not, whether or not it is running as administrator, and whether or not the UAC is enabled. The value of the 'bit' attribute indicates whether the operating system is 32-bit (32) or 64-bit (64).

In the second part, the content in 'CDATA' is the information about the installed software. The malware enumerates all the subkeys of HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/Uninstall and gets the value of their key name, 'DisplayName' and 'DisplayVersion'. It will compose a string with the format 'DisplayName_value (DisplayVersion_value)' and connects every subkey's string with the character ';'. It should be noted that the malware only recognizes English characters, so it changes the non-English characters to '?'. The malware attaches a string, 'Starting path: %d', to the end of the connected string. Despite what its name may suggest, the content of 'Starting path' is not a real path of the malware. Instead, it is a figure indicated in the MIC (Mandatory Integrity Control [2]) level of the path which the malware located. There are seven levels: untrusted, low, medium, medium plus, high, system and protect process, which correspond to the values 1–7. If the operating system version is higher than or equal to Windows NT 6.0, the malware uses the GetSidSubAuthority API to get the MIC level. Otherwise, it sets the figure to 5. Figure 7 shows the raw data which will be sent to the server.

```

<loader><get_module unique="E4230720?F5E422_430ec6a2af3ad9939cc3009cd619e338"
botnet="200" system="23120" name="bot" bit="32"/><soft><![CDATA[Windows XP
???? - KB873333 (20050114.005213);Windows XP ????? - KB873339 (20041117.092459
);Windows XP ????? - KB885250 (20050118.202711);Windows XP ????? - KB885626 (
20040909.122822);Windows XP ????? - KB885835 (20041027.181713);Windows XP
???? - KB885836 (20041028.173203);Windows XP ????? - KB886185 (20041021.090540
);Windows XP ????? - KB886677 (20041015.135028);Windows XP ????? - KB887472 (
20041014.162858);Windows XP ????? - KB887742 (20041103.095002);Windows XP
???? - KB887797 (20041018.133824);Windows XP ????? - KB888113 (20041116.131036
);Windows XP ????? - KB888302 (20041207.111426);Windows Media Player 10 Hotfix
- KB888656;Windows XP ??? (KB890046) (1);Windows XP ????? - KB890859 (1)
;Windows XP ????? - KB891781 (20050110.165439);Windows XP ??? (KB893066) (2)
;Windows XP ????? - KB893086 (1);Windows XP ??? (KB893756) (1);Windows
Installer 3.1 (KB893803);Windows XP ?? (KB894391) (1);Windows XP ??? (KB896358
) (1);Windows XP ??? (KB896422) (1);Windows XP ??? (KB896423) (1);Windows XP
??? (KB896424) (1);Windows XP ??? (KB896428) (1);Windows XP ??? (KB896688) (
1);Windows XP ?? (KB896727) (1);Windows XP ?? (KB898461) (1);Windows XP ??? (
KB899587) (1);Windows XP ??? (KB899588) (1);Windows XP ??? (KB899589) (1)
;Windows XP ??? (KB899591) (1);Windows XP ??? (KB900725) (1);Windows XP ?? (
KB900930) (1);Windows XP ??? (KB901017) (1);Windows XP ??? (KB901190) (1)
;Windows XP ??? (KB901214) (1);Windows XP ??? (KB902400) (1);Windows XP ???
(KB904706) (1);Windows XP ??? (KB905414) (1);Windows XP ??? (KB905749) (1)
;Windows XP ??? (KB905915) (1);Hotfix for Windows Media Player 10 (KB907658)
;Windows XP ??? (KB908519) (1);Windows XP ?? (KB910437) (1);Windows Media
Player (KB911564) ???;Windows Media Player 10 (KB911565) ???;Windows XP ???
(KB911927) (1);Windows XP ??? (KB912919) (1);Windows XP ??? (KB913446) (1)
;Windows Genuine Advantage Validation Tool;Windows Media Format
Runtime;Windows Media Player 10;WinPcap 4.1.1 (4.1.0.1753);WinRAR
archiver;Wireshark 1.2.9 (1.2.9);WebFldrs XP (9.50.7523);UltraEdit-32 (13.00a)
;SSH Secure Shell;Winalysis (3.0.1);Debugging Tools for Windows (6.8.4.0)
;Microsoft Windows XP CD ??? HighMAT ?? (1.1.1905.1);VMware Tools (8.4.5.14951
);Microsoft Visual C++ 2008 Redistributable - x86 9.0.21022 (9.0.21022);???
StyleXP ??? N ? 1 ??? 8.0 (8.0);Starting path: 5]]></soft></loader>

```

Figure 7: The raw data sent to the server.

Finally, the malware gets a random DWORD key and uses a XOR operation to encode the raw data of every DWORD.

COMMUNICATION

Before the communication begins, the malware will parse the server data (Figure 5). The parsing function checks the special characters (e.g. '://', '@', '/', ':', '?', '#') to locate the communication protocol, server address, file path, arguments, port, user name and password. If the hard-coded URL does not have a communication protocol, the malware will set 'HTTP' as default. The port field also has default values: for HTTP it is 80, for HTTPS it is 443 and for FTP it is 21. Other fields default to NULL if no matching value is found in the string. In this sample, the server data is very simple, with only server address and port. As shown in Figure 5, the server URL is of the format 194.28.87.125:4443. By design, the malware uses HTTPS for communication, so before calling the parsing function, the malware will prepend the URL string with the HTTPS protocol. After calling the

parsing function, the malware will get the server address as 194.28.87.125 and the port as 4443. After parsing, the malware uses the InternetConnectW API to connect to the server, sends the encrypted data using the HttpSendRequestW API, and finally reads the response from the server using the InternetReadFile API.

The data received from the server is also encrypted. The first four bytes is a DWORD key which is used as the XOR key to decode the data after it by DWORD (Figure 8).

The decoded data is encased in XML code which starts with a '<root></root>' element. In the root node, there are two sub nodes, <nodes></nodes> and <module name="bot" bit="32"></module>. The content in the 'module' node is encoded with the BASE64 algorithm. Figure 9 shows a piece of the data after decoding.

The first 0x80 bytes of the decoded data is junk code, after which is a DLL. The malware writes this DLL into a TEMP file whose directory is the same as the malware. Then it creates a registry entry: HKEY_CURRENT_USER/Software/

00000000	45 2E DB 1A 79 5C B4 75	31 10 E7 74 2A 4A BE 69	E. ?y~碼1.鏡*J總	00000000	3C 72 6F 6F	74 3E 3C 6E	6F 64 65 73 3E 64 45 79	<root><nodes>dEy
00000010	7B 4A 9E 63 73 41 89 63	03 6A B6 2B 0B 4C B7 59	{J潘sA塩.j?.L殺	00000010	36 6F 52 79 46 44 6D 31	4E 62 6C 43 52 4E 4D 38	6oRyFDm1Nb1CRNM8	
00000020	17 60 96 22 3D 67 B2 72	20 62 EF 4D 10 60 8E 72	. ?=g膜 b影. 濃	00000020	78 49 69 68 65 4C 34 57	55 4E 55 68 43 6F 57 4D	xIiheL4wUNhCoWM	
00000030	06 41 8C 57 3C 60 A2 77	33 5C BA 77 21 68 B5 7C	.A學< 3>婁!h妹	00000030	79 4E 79 6D 76 72 51 6D	64 46 6E 66 39 68 53 73	yNymvrQmdFnf9hSs	
00000040	7C 46 88 69 2B 61 BE 7D	12 5D ED 4C 7D 4B 97 74	F坐+a織.}鞣}K裡	00000040	6E 4F 65 67 57 73 36 56	38 65 4C 6E 31 39 38 38	nOegWs6V8eLn1988	
00000050	74 17 E3 22 70 17 AD 7C	33 4A 93 42 03 61 83 43	.t. ?p. 驟3J揃.a備	00000050	35 39 76 66 76 64 48 58	46 4F 58 59 72 68 74 57	59vfvdHXFOXYrhtW	
00000060	37 46 AF 4D 0B 1F 9D 56	0B 54 A2 6A 6A 54 EF 4E	7F疾. .漉.I じ風	00000060	4E 31 46 4C 4E 7A 79 70	2F 7A 34 54 63 67 77 2F	NIFLNzyp/z4Tcgw/	
00000070	26 49 AC 35 6E 05 ED 56	26 48 8C 23 01 1E 83 6D	&I7n. 變GH?. 僮	00000070	2B 2F 36 4C 63 66 57 39	44 30 58 77 62 75 32 75	++6LcfW9D0Xwbu2u	
00000080	27 5B E9 6F 2D 7A A2 79	35 54 A3 70 6A 5E EA 2F	[間-z 5T j^?	00000080	68 54 79 63 70 7A 78 6A	2F 70 31 35 4E 4B 79 4F	hTycpzzx/jp15KNy0	
00000090	0E 60 A2 55 24 58 E8 52	0D 48 AB 57 37 74 9D 2C	. ' sX鏢.H劫7t?	00000090	61 76 33 48 48 66 70 4D	72 5A 46 36 69 65 67 52	av3HHfpMrZF6iegR	
000000A0	2C 4B BC 48 2E 58 B3 49	3F 5E AD 79 12 6C B4 2A	.K輕.X呀?^璫. 1?	000000A0	6B 76 68 53 7A 70 76 63	57 42 6F 30 53 31 37 76	kvhSzpvcWBo0S17v	
000000B0	16 1F EC 6C 09 68 AD 2A	2E 76 9D 59 29 19 97 5E	. .航.h?.v縱). 棕	000000B0	4C 46 76 30 6B 58 46 43	6C 37 4C 47 35 6D 65	LFvDk3FC17LD7ume	
000000C0	72 5B B6 7F 71 05 ED 42	36 5D 8B 6F 29 19 EE 28	r[?q. 糖6}績). ?	000000C0	34 2B 36 58 73 73 50 75	6C 37 35 32} 72 35 4F 4A	4+6XssPu1752r50J	
000000D0	37 1B 94 50 3C 57 ED 71	1D 41 96 74 10 4A AB 2E	7. 擡<w鞍.A坐.J?	000000D0	79 79 36 6B 58 6F 4D 6E	55 64 70 34 57 6B 52 32	yy6kXoMnUdp4WkR2	
000000E0	12 45 89 28 34 7F EE 7C	29 7D 9E 4E 13 61 ED 2F	.E?4!願.)酒.a?	000000E0	71 51 35 66 6C 53 45 54	56 4F 36 35 5A 44 55 4C	qQ5f1SETV065ZDUL	
000000F0	1F 6A 8E 56 11 59 91 60	16 78 AC 56 20 16 9D 22	.j峽.Y備.x理. 1?	000000F0	54 77 4A 7A 53 56 77 4C	65 38 46 38 49 58 42 61	TwJzSVwLe8F8IXBa	
00000100	0C 76 99 7B 0A 67 ED 60	77 1A 83 55 0F 42 AC 4A	.v檔.g驪w.價. B這	00000100	4F 49 36 7A 32 34 58 4F	4A 6C 77 50} 67 42 6D 2B	O16z24X0J1wPgBm+	

Figure 8: XOR the downloaded data.

00000000	2A 6E 9D 87 A5 12 5D 3C	F7 D6 67 8F 8F D7 74 B8	*n澈?]<鬚g寄譬?
00000010	A7 7D FC 9D B6 0C C1 B3	B5 F5 22 3F C6 FB 44 14	詩?臉吊"?汽D.
00000020	74 DE 13 B9 32 AC 60 FB	B6 49 BC 81 07 30 C8 36	t??瓠 I紛.0?
00000030	76 9E 76 4C 38 80 AE CA	73 71 D2 B0 8B 41 BA 5C	v濶L8E鄭sq野端筆
00000040	9F 7A 05 81 C9 BF A7 3B	B8 D5 5B FE A8 05 19 DE	莞. 伉伽; 刚[. .?
00000050	AA 7E BA 4C 65 BD 39 3C	6A 70 6C 23 FB D1 CD 7D	猛簪e?<jp1# 蚧
00000060	2F 90 73 8A E2 61 5E F4	90 CC 6B D7 EC 1E 1E 9E	/恠媿a^毓濼嘴..?
00000070	92 37 E1 05 D7 5D 13 94	34 BE E3 51 8E 7D BF 59	??諗.?俱Q袷縹
00000080	4D 5A 90 00 03 00 00 00	04 00 00 00 FF FF 00 00	MZ?..... ..
00000090	B8 00 00 00 00 00 00 00	04 00 00 00 00 00 00 00	?.....@.....
000000A0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00?..
000000B0	00 00 00 00 00 00 00 00	00 00 00 00 F8 00 00 00?..
000000C0	0E 1F BA 0E 00 B4 09 CD	21 B8 01 4C CD 21 54 68	..?.???L?Th
000000D0	69 73 20 70 72 6F 67 72	61 6D 20 63 61 6E 6E 6F	is program canno
000000E0	74 20 62 65 20 72 75 6E	20 69 6E 20 44 4F 53 20	t be run in DOS
000000F0	6D 6F 64 65 2E 0D 0D 0A	24 00 00 00 00 00 00 00	mode....\$.

Figure 9: A piece of data decoded with BASE64.

Microsoft/Windows/CurrentVersion/Explorer/CLSID/%s/ShellFolder. The '%s' is a GUID which is transformed from the MD5 of a variant of the System_Info data block (described earlier). The System_Info variant only adds a byte, 0x13, following the System_Info block. The value of this registry is encrypted by a customized algorithm. Its raw data is in the format <cfg net="%d" build="0"><startup>%s</startup>%S</cfg>. The value of the 'net' attribute is botnet_id, the content in the 'startup' section is retrieved from the 'nodes' section, which is sent from the C&C server, the content of the 'del' section is the path of the malware. Finally, the malware calls the CreateProcessW API to run the DLL with argument 'rundll32.exe "<DLL_path>" NotifierInit'. The DLL has an export function, NotifierInit, and this DLL will carry out further orders received from the server.

CONCLUSION

By analysing the malware in detail, we have learned about its working mechanism and how it gathers information and communicates with the C&C server. We can now forge data and send it to the server, decode the response and check the server commands. In this way, we might obtain more

commands for further research or obtain the latest variants in order to keep track of this malware.

REFERENCES

- [1] http://ibsensoftware.com/products_aPLib.html.
- [2] http://en.wikipedia.org/wiki/Mandatory_Integrity_Control.

Editor: Martijn Grooten
Chief of Operations: John Hawes
Security Test Engineers: Scott James, Tony Oliveira, Adrian Luca
Sales Executive: Allison Sketchley
Editorial Assistant: Helen Martin
Developer: Lian Sebe
Consultant Technical Editor: Dr Morton Swimmer
 © 2015 Virus Bulletin Ltd, The Pentagon, Abingdon Science Park, Abingdon, Oxfordshire OX14 3YP, England.
 Tel: +44 (0)1235 555139. Fax: +44 (0)1865 543153
 Email: editorial@virusbtn.com
 Web: <http://www.virusbtn.com/>