SoBig-G

(Addressing the weaknesses in the design of SoBig-F)

Revision: 1.12

Arrigo Triulzi*

4th September 2003

Abstract

In this paper we discuss a few ideas for a possible evolution in the design of the SoBig virus. In particular we suggest a mechanism for avoiding the weaknesses in the current design in a corporate environment which are linked to the restricted host list and need to synchronise time using a "visible" protocol. We also address the issue of better "blending" in a corporate environment to delay discovery and maximise effectiveness.

1 Introduction

The SoBig-F virus is the sixth variant of a mass-mailing virus which uses executable attachments (PIF and SCR files) to infect Windows-based systems. It started spreading around 19th August 2003 and is designed to deactivate on 10th September 2003. Once a host is infected the virus uses its own SMTP engine to further propagate via addresses found in the Outlook address-book via MAPI. The virus then uses an internal list of NTP servers to to keep its clock synchronised and update its code by contacting a pre-defined encrypted list of hosts when a specific time stamp condition is met. These hosts would provide a URL from which to download the upgrade rather than the upgrade itself. A more detailed description of SoBig-F is available in [1, 2].

The aim of this paper is to discuss a number of weaknesses in the design of SoBig-F with particular attention to methods which could be used to make its detection substantially harder in a corporate environment.

2 Weaknesses in the original design

Once the virus was discovered in the wild and disassembled a number of interesting features became apparent:

- the virus does not make use of local information with the exception of the address book (and this it does "passively" by executing when the attachment is clicked),
- the virus would keep an accurate time by synchronising to NTP servers from an internal list of twenty hosts[1],

*e-mail: arrigo@alchemistowl.org, 'phone: +41 76 453 1649

[1] See [5] for a definition of the NTP protocol.

- at any time between 19:00 and 22:00 UTC on Friday and Saturday the virus would attempt to "update" automatically by fetching a URL from a set of "master" hosts. It would then fetch and execute the contents of the URL and, finally,
- the list of twenty "master hosts" was encrypted[2].

This last "feature" caused a race to decrypt the list and disable the twenty masters before 19:00 UTC on 24th August 2003 (the first Friday after the spread of the virus had begun). This was achieved by the F-Secure team (see [1, 7, 8] for a chronology of the events).

It is surprising to note that this virus, as many others before it, did not make use of any local settings on the system. This should undoubtedly raise its profile in a well-monitored network, in particular the URL fetching and subsequent update should trigger firewall filters on outbound connections not via the web proxy service.

Furthermore the NTP list meant that by monitoring unusual accesses to UDP port 123 to hosts on the list one could check the spread or indeed the presence of infected hosts in their network.

Finally the "master list" of systems from which the update URL was to be obtained, once decrypted, meant that the whole virus update mechanism was impaired by shutting down the twenty hosts.

3 Addressing the weaknesses

We now address the weaknesses discussed in the previous section and propose a different mechanism to achieve the same goals. In this respect we are not modifying the, as yet unknown, aim of SoBig-F inasmuch as improve the mechanisms it uses to achieve the goal of downloading the update code.

In a corporate environment, especially one with proper perimeter protection and monitoring, we need to ensure that we minimise the chances of our position being discovered. Furthermore we need to maximise the probability of the updates being successful by making it difficult to block the process.

There are three areas which need to be addressed separately: localised settings, time updating and triggering and finally the update process.

3.1 Using local settings

We should first address the need to "blend" into the corporate environment to minimise the chances of detection.

This can be achieved by implementing the following suggestions:

1. Once a system is infected the virus should detect as many settings as possible. For example:
   - SMTP relays,

[2]Note that the "master hosts" were completely different systems from the publicly available NTP servers chosen to synchronise the time.

- HTTP proxies,
- NTP servers and local timezone.

These should then be used by the code rather than direct access,

2. In particular, if web access is required and web proxy settings are used, attention should be paid to proxy authentication. As this is often saved by the user to avoid having to type it at each login using the local proxy settings directly ensures that it is picked up and access to the web is then open,

3. The local anti-virus program should be monitored for updates and a suitable mutation triggered when an update request is detected to minimise the possibilities of a new signature detecting it,

4. avoid disabling the anti-virus program or indeed any other protection mechanism unless absolutely essential,

5. When requesting web information pick a "smart" user agent. Most fields can be constructed from information available on the system or alternatively sniffing the first few TCP packets being transmitted from the host to port 80, 3128 or 8080 and picking out the relevant HTTP header.

All of the above are designed to ensure that the virus remains "under the radar" as much as possible. In particular with respect to web accesses the chances of the web proxy logs being read carefully is minimal and "cyber nanny" software will not be bypassed (thereby possibly triggering an error in the logs). As we shall discuss later, judicious choice of URLs will not trigger either proxy-bypass blocking or cyber nanny software.

3.2 Time synchronisation and triggering

Fully-fledged NTP seems to be a bit of an overkill for the purposes of the virus: 9:00 UTC and 9:01 UTC are close enough to unleash a sufficiently large DDoS storm or whatever "payload" the virus is designed to carry.

Using port 123 from a host which normally does not synchronise time remotely or uses a local time source (e.g. SNTP under Windows 2000) clearly gives away the position of the virus too easily.

In most corporate environments and well-designed networks NTP is cascaded through a set of tiers: at the top tier is either a local or external GPS-synchronised clock which is often propagated to the routers which then act as tier-2 servers for all the hosts in the network. By synchronising systems with the topologically closest router a good time distribution and synchronisation can be achieved in a network. Systems running Windows would, in particular, often use the SNTP protocol to maintain a good quality local clock.

The only systems which would have external access to UDP port 123 would be either tier-2 NTP servers using off-site tier-1 NTP servers to synchronise their clocks or specific systems which required off-site NTP access. The firewalls should therefore block or drop *and log* off-site UDP requests to the master servers. Hence a simple scan of the logs for disallowed accesses to UDP port 123 would reveal the infected hosts (or at the very least hosts which do not adhere to the company policy).

3.2.1 Web-reachable precise time sources

Why not use a publicly accessible web site which gives a very precise time in an easily parseable format? Using the system's web proxy settings ensures that this would be comfortably under the radar and a request every six hours should be sufficient to correct most PC clock drifts to within a few minutes of the true time. An excellent candidate is the "US Naval Observatory Master Clock Time"[3]:

http://tycho.usno.navy.mil/cgi-bin/timer.pl.

The output, presented in its HTML source is absolutely perfect for the purposes of coarse time synchronisation (abridged, long lines truncated and continued on the next line):

```
<title>What time is it?</title>
<h2> US Naval Observatory Master Clock Time</h2> <h3>
<br>August 25, 15:39:06 UTC
<br>August 25, 11:39:06 AM E
<br>August 25, 10:39:06 AM C
<br>August 25, 09:39:06 AM M
<br>August 25, 08:39:06 AM P
<br>August 25, 07:39:06 AM Y
<br>August 25, 05:39:06 AM A
</h3></b><p><a HREF="http://tycho.usno.navy.mil">Time Service \
 Department, US Naval Observatory</a>
```

We also need to consider access via a web proxy. Most web proxy software does not cache CGI binaries in its default configuration, alternatively a simple reload will provide a solution. To detect the need for a reload it is sufficient to compare the last time stamp with the new one: if the deviation is too small compared with the locally elapsed time then a fresh copy is needed (this also ensures that there is a certain degree of correctness in the downloaded page).

3.2.2 Using a local web server for the time

Some sites are very restrictive in the list of external web sites which are accessible from the internal company network so access to external web sites for "time over HTTP" might not be possible.

In this particular scenario an alternative is nevertheless possible: simply make use of HTTP headers to obtain the time. For example, if we look at the output of the HTTP "HEAD" method we noticed that we have an easily parseable time stamp in the form of the "Date" field in the server response headers:

```
> HEAD / HTTP/1.1
> Host: www.alchemistowl.org
> Accept: *
>
```

[3] This site used to be known as the "Directorate of Time".

```
< HTTP/1.1 200 OK
< Date: Thu, 04 Sep 2003 13:09:18 GMT
< Server: Apache/1.3.27
[...]
```

The virus can then obtain the current time by requesting the default starting page as set for the local browser configuration. It would be advisable to, once again, use local settings for the "User Agent" field and make use of the "GET" method to make the access look legitimate in the web server logs.

3.2.3 Triggering

Furthermore we need to consider the "triggering". Is it a good idea to trigger between 19:00 and 22:00 UTC? That might work reasonably well in continental Europe but most definitely not in the USA or Asia. The triggering should be localised (in particular in view of the next section which makes the upgrades "globalised") to minimise impact on the network and the reaction by the IT staff. This information should be gathered and used as discussed in § 3.1.

3.3 Downloading "upgrades"

We now have a mechanism to obtain a sufficiently precise time stamp and an improved triggering point for the upgrade so we can now consider the upgrade procedure in detail.

The aim of the upgrade procedure is to be able to obtain new code or operational instructions from remote sites and to do so in a way which will minimise the chances of the "upgrade net" being shutdown.

There is a simple and yet powerful mechanism to achieve this aim: build upon Google's infrastructure (see [3]). Google's network of search engines is extremely well connected from any location of the Internet and is resilient but, even more usefully, offers a "cache" facility whereby the web pages which match the search terms are held on Google systems saving the need to reach the original web site.

3.3.1 Part I: seeding Google

To make use of the Google infrastructure we need to "seed" it appropriately with our data. We begin by designing a suitable page which contains the upgrade code and specific values for the "keywords" HTML meta tag. These pages are placed on the Internet making use of as many different providers as possible (both free and non-free). We then submit them to Google via their "add URL" web pages.

The process to be followed to place our upgrade code throughout the Google infrastructure is simple, for each created web page:

1. the "keywords" meta tag in the HTML head contains a unique pair of prime numbers numbers, (p_i, p_j) where $p_i, p_j \in \{1, \ldots, 2^{16}\}$ and we disallow (p_j, p_i) as a separate pair to guarantee uniqueness. This is placed in the HTML head as follows:

   <head><meta name="keywords" content="p_i p_j"/></head>

The total number of possible web pages is given by

$$\frac{1}{2}\binom{\pi(2^{16})}{2} \approx \frac{1}{2}\binom{5909}{2} = 17455186 \approx 17 \text{ million},$$

where $\pi()$ is defined in Tchebychef's Theorem (see [4]) to be an approximation to the number of prime numbers less than its argument. We need to divide by two as we disallow the same prime pair with the primes switched.

2. the body of the web page contains the upgrade code encoded in a suitable format. It should be in the simplest possible form so that the decoding engine in the payload need not be too complicated. The content could be encrypted if need be but experience has shown that since you need to carry the decryption algorithm and key with the virus this would not pose much of a challenge. It could be argued that the virus code could use itself as a one-time pad to decrypt the upgraded code (on the assumption that the lengths are made match or the upgrade code is shorter) as the optimal encryption method.

3. each of the pages is then "seeded" into Google via the appropriate web page:

   `http://www.google.com/addurl.html`.

   According to the documentation on the web site and empirical evidence the seeded pages will normally be visited within a week of submission by the Google "web spider". The form submission can be trivially automated in a few lines of Perl but care should be taken to minimise the rate of submission to remain well beneath what Google would consider a mass submission attempt.

By repeating the above for as many web pages as possible we guarantee that the search for a web page having as keywords any two primes less than 2^{16} will succeed.

There is no need to seed each and every one of the 17 million possible web pages as long as the viral code is suitably amended to ignore searches returning no hits. The advantage of not seeding each and every one of the 17 million possible page is that just like the virus needs to search for a valid prime combination so does the anti-virus community making the disabling of the upgrade procedure more time-consuming.

Note that using small primes does pose the risk that other pages might match the search criteria in Google. This can be prevented by selecting a suitable different interval, for example primes in the interval $(2^{16}, 2^{17})$ of which there are approximately 5200.

3.3.2 Part II: retrieving the upgrade

By monitoring the web access logs of some of the seeding sites[4] we can have a relatively good idea of the date after which the seeding pages are present in the Google database and caches. From our seeding procedure we also know that the pages can be accessed through the unique pair of prime numbers used as the identifying keywords. At which point the virus can be deployed and the update procedure triggered making use of the infrastructure we have prepared.

[4]The use of Geocities and other free services exclusively would make this impossible so it is important to have at least one site under complete control.

The idea for the upgrade procedure comes from a game called "Google Whacking" (see [6]) which consists in finding two unrelated words which give exactly one search result from Google. We simply "cheat" and via the seeding process make it extremely likely that there is exactly one search result when we try our queries.

The virus now follows the following procedure to download the upgrade:

1. pick two random prime numbers p_i and p_j such that $p_i, p_j \in \{1, \ldots, 2^{16}\}$,
2. search Google, perhaps via the local web proxy, simply requesting the appropriate URL given by:

   `http://www.google.com/search?&q=%22p_i%20p_j%22`

   (where `p_i` and `p_j` correspond to the p_i and p_j chosen in step 1).

   Note in particular that because of our seeding procedure we can even consider using the "I'm Feeling Lucky" button:

   `http://www.google.com/search?&q=%22p_i%20p_j%22&btnI=I'm+Feeling+Lucky`

   With this last link we can be connected directly to our upgrade code.
3. download the upgrade code from the link.
4. install the upgrade or execute whatever is expected from it.

There is an even better variant, at point 2 above we actually don't need the original website but instead pick the "cache" link. This prevents the "I'm Feeling Lucky" shortcut but ensures that the best possible connectivity is obtained and hence part 3 is now direct from Google. It also makes it much harder to remove the upgrade code as it is stored in all the Google cache systems.

Note that in a corporate environment the local proxy server could already be caching the page, making it even harder to notice infected sites but the design of the update algorithm is such that this likelihood exists only for large number of updates as the choice of prime number pairs is random.

Finally, another variant is to use the Google groups infrastructure (`groups.google.com`) instead of the web infrastructure to store the upgrades and then accessing them via the article number but this exposes the possibility of "cancels" issued by the NNTP infrastructure. It should also be noted that NNTP access is not necessarily available in most companies.

4 Conclusions

There are very few obvious mechanisms which would impair the functionality of this virus. One of the entry points into the virus is Google's URL adding mechanism: if new URLs are carefully monitored before being crawled then perhaps the update mechanism could be defeated. This would imply quite a bit of intelligence in detecting that the seeded pages are viral update code but then superficial analysis could be thwarted by using steganographic encoding in, for example, JPEG images.

Once the virus has been distributed the upgrade infrastructure is a point that can be targeted to thwart the virus just like SoBig-F. As the upgrade relies on the uniqueness of the search results a check can be made on search results which give rise to "Google Whacks". When a "Google Whack" takes place the result can be made non-unique by adding another page which will remove the uniqueness of the search result. The new page can be very similar to the original so that it is virtually indistinguishable from a parsing point of view. At which point the virus will first of all have to choose between two results which might already cause it to fail or download the wrong page and crash when trying to upgrade itself with its contents.

Of course the above protection mechanism has the limitation that one still has to figure out the algorithm, if one exists, used to choose the terms for the search. This requires the disassembly of the virus, as always, and can be made sufficiently arduous that it is not completed before the trigger time stamp or some other event takes place (e.g. wiping the contents of a hard-disk). At which point the onslaught becomes inevitable.

Acknowledgements

The author wishes to thank Dr. Diana Bosio for suggesting prime numbers for "Google Whacking", Jose Nazario for pointing out the time in HTTP response headers and Alessio Bragadini for pointing out that most web proxy servers do not cache CGI scripts in their default setup.

References

[1] F-Secure Computer Virus Information Pages: Sobig.F. F-Secure Corporation, August 2003. http://www.f-secure.com/v-descs/sobig_f.shtml.

[2] Symantec Security Response: W32.SoBig.F@mm. Symantec Corporation, August 2003. http://securityresponse.symantec.com/avcenter/venc/data/w32.sobig.f@mm.html.

[3] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, March-April 2003. http://csdl.computer.org/dl/mags/mi/2003/02/m2022.pdf.

[4] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 5th edition, 2000.

[5] D. Mills. Network Time Protocol (Version 3): Specification, Implementation and Analysis. IETF, March 1992. http://www.faqs.org/rfcs/rfc1305.html.

[6] G. Stock. googlewhack: The Search for "The One", 2002. http://www.googlewhack.com/.

[7] Johannes Ullrich. Handlers Diary August 22nd 2003. SANS Internet Storm Center, August 2003. http://isc.sans.org/diary.html?date=2003-08-22.

[8] Johannes Ullrich. Handlers Diary August 26th 2003. SANS Internet Storm Center, August 2003. http://isc.sans.org/diary.html?date=2003-08-26.