NOVEMBER 2013

virus BULLETIN

Covering the global threat landscape

CONTENTS

IN THIS ISSUE

GERMAN GATHERING

The first week of October saw the 23rd anniversary of German reunification and the 23rd Virus Bulletin International Conference – in Berlin. Helen Martin reports on the latter.

page 4

DEGENERATION GAME

When one has a nice idea – such as a tricky method for encoding data – it is common to take that idea and improve on it. It is rare to see someone take such an idea and degenerate it, but that's exactly what we see in W32/Tussie.B. Peter Ferrie reports.

page 9

NEW BOT ON THE BLOCK

Neurevt is a relatively new HTTP bot that already has a lot of functionalities along with an extendable and flexible infrastructure. Zhongchun Huo takes a detailed look at its infrastructure, communication protocol and encryption scheme.

page 11

ISSN 1749-7027

'If anyone were to invent SMTP today and decide it was a good idea for messages to be sent in plain text, they would receive short shrift.'

Martijn Grooten, Virus Bulletin

PERHAPS EMAIL IS BROKEN AFTER ALL

It has long been popular in anti-spam circles to make bold claims about email being horribly broken. After all, with at times up to 90 per cent of all email traffic being spam – a lot of it carrying malicious payloads – how can there *not* be something fundamentally wrong with it?

I have always maintained that this is a flawed argument. Spam, or the sending of unsolicited bulk email, is an explicit feature of email – combining the ability of computers to do many things in a very short period of time, with the allowance we see in the regular postal system for the sending of unsolicited mail. Spam is a feature of email, rather than a bug.

Even if spam were to become so out of hand that we needed to replace the email protocol, its replacement shouldn't just generate less spam than we are dealing with now. Rather, it should reduce the amount of spam that users actually receive – which, thanks to today's spam filters, remains relatively low. That's a huge threshold.

But recently there have been a couple of incidents that have left me wondering if email *does* need to be fixed after all.

First to make me think again about the state of email were the much-discussed revelations surrounding the NSA. We now know that agencies like the NSA in the US and GCHQ in the UK tap all the Internet traffic that passes through their respective countries. The cunning ways in which they break or backdoor cryptography have been well documented – but for most email, neither cunning nor cheating is required: the messages are sent over the wires in plain text.

Of course, we didn't need Edward Snowden to point this out, and anyone who believed that secret agencies (and not just the NSA and GCHQ) *wouldn't* tap the cables they had access to, was rather naïve. But Snowden's revelations did act as a wake-up call for the email community.

If anyone were to invent SMTP today and decide it was a good idea for messages to be sent in plain text, they would receive short shrift from the Bruce Schneiers and Matthew Greens of this world. And rightly so.

Of course, encrypting the email when it is in transit would still leave opportunities for powerful entities to read the unencrypted messages. For those messages where secrecy is important, end-to-end encryption mechanisms such as PGP should be used. And for those messages where secrecy is of vital importance, and where metadata should remain secret, email might not be the right protocol at all.

The second event that led me to wonder whether email is broken was the recent DNS hijack suffered by some prominent security companies. The hijack resulted in the companies' websites appearing to have been hacked – but in some cases it also resulted in the companies' emails being redirected to a different server.

There is nothing in standard SMTP to prevent emails being sent to the hackers who have taken control of the DNS. Worse, email authentication systems such as DKIM and SPF 'helpfully' provide mechanisms that would have allowed the hackers to send emails on the companies' behalf and which would have been accepted by a lot of spam filters as legitimate traffic. We should be grateful that in this case the hackers chose to focus their attention on the companies' websites rather than starting what could have been a damaging phishing campaign.

The two turns of event mentioned here highlight the fact that the email community needs to prioritize its efforts to make sure that SMTP transactions that are sent over the public Internet are both encrypted and authenticated. This shouldn't be presented as a way to make email more secure – it won't stop the most powerful attackers from reading our emails. However, it would enable email to meet what should be one of our basic expectations of a digital mail service.

We've been doing rather well in our fight against spam. Now let's tackle this issue too.

NEWS

GOVERNMENTS SEEK TO BOOST CYBER DEFENCES & SKILLS

Recruitment began last month for the UK's Joint Cyber Reserve Unit, which will work alongside the UK's regular Armed Forces 'to protect critical computer networks and safeguard vital data'.

Under the £500m initiative, the Ministry of Defence (MOD) plans to recruit hundreds of computer experts as cyber reservists, with the recruitment drive targeting three sectors: regular personnel leaving the Armed Forces, current and former reservists with the requisite skills, and individuals with no previous military experience, but who have the technical knowledge, skills, experience and aptitude to work in the area of cybersecurity.

The head of the fledgling unit, Lieutenant Colonel Michael White, caused some eyebrows to be raised last month when, appearing on *BBC* news and current affairs programme *Newsnight*, he said that he would be open to hiring criminally convicted hackers with the relevant expertise – providing they could get through the stringent security clearance process.

The anti-malware industry has historically taken a very dim view of former cybercriminals being employed by security firms, but in this case experts have cautiously been more accepting of the idea – presumably on the assumption that the MOD's vetting process would weed out any individuals still harbouring more nefarious motivations.

In February this year, the UK's National Audit Office warned that the UK faced a current and future cybersecurity skills gap – saying that it could potentially take up to 20 years to address the gap. With such a shortage of skills it is perhaps little surprise that the MOD is willing to entertain the idea of hiring reservists with spent criminal convictions.

A number of initiatives are already running in the UK in an attempt to increase interest and involvement in the cybersecurity arena: the nation's first Cyber Academy was launched in September by non-profit organization *e-skills UK*, and another non-profit organization, *Cyber Security Challenge UK*, has created a lesson plan which offers schools a series of fun and engaging activities to help children gain a better understanding of cybersecurity matters and inspire them to consider a career in cyber defence. *e-skills UK* is also developing the first nationally available degree-level apprenticeships in cybersecurity.

Of course, the UK is not the only country seeking to boost its cyber defences. The Indian government, for one, is also taking steps to increase its cybersecurity skills. The National Security Database (an offshoot of the country's Information Sharing and Analysis Center [ISAC]) has pledged to increase the number of reverse engineering professionals in the country from fewer than 5,000 currently to 100,000 by 2015. The NSD has already developed a series of in-depth reverse engineering bootcamps which aim to help engineers understand different aspects of application security, learn anti-cracking techniques and learn how to create secure code for internal use.

FINNISH GOVERNMENT BREACH

In somewhat related news, it emerged last month that the Finnish Ministry of Foreign Affairs had been the victim of a malware attack over a period of four years. According to Finnish TV channel *MTV3*, the breach was discovered earlier this year, and the malware – which was described as being 'similar to Red October' – targeted communication between Finland and the European Union.

It is suspected that Russian or Chinese intelligence agencies were behind the attack – and that Finland was not the only country affected.

Finland approved an official national cybersecurity strategy in January this year, with Phase 1 of the strategy due to be put into practice during 2013–2015 and the entire set of measures to be implemented by 2016.

EMPLOYEE AWARENESS AND SECURITY BUDGETS STILL FOUND LACKING

Only 17% of respondents in *Ernst & Young* (*EY*)'s 16th annual global information security survey said they felt that their company's information security function fully meets the needs of their organization. More than 1,900 information security executives across 25 industry sectors and in 64 countries took part in the survey, which was conducted between June and July this year.

While some of the survey's findings were encouraging – with many organizations seeing increased investment in information security – 65% of respondents still cited budget constraints as the biggest obstacle to delivering value to the business, and 50% of respondents said they felt that a lack of skilled resources stood in the way of addressing their organization's security needs.

With phishing and targeted attacks rife, ongoing security awareness training is one way organizations can improve the chances of employees spotting social engineering attacks – so it was a surprise that only 23% of respondents ranked employee security awareness training as a high priority, with an even more surprising 32% of respondents ranking it as their lowest priority.

The survey's authors concluded that organizations need to place more emphasis on improving employee awareness, increasing budgets and devoting more resources to innovating security solutions.

CONFERENCE REPORT

VB2013: BERLIN TIME

Helen Martin

With high-tech industries, renowned universities and research facilities nestling alongside historic sites, diverse architecture, a dynamic arts scene and a lively nightlife, the vibrant city of Berlin combines the old and the new in a way that few other places manage, and it caters to all tastes and interests. The same could be said for the 23rd Virus Bulletin International Conference (VB2013) – which really did seem to have something for everyone.

The Maritim Hotel Berlin, which played host to VB2013, stands directly opposite the historic Bendlerblock building complex (which now houses the German Resistance Memorial Centre), a couple of strides away from the Tiergarten (the largest and oldest public park in Berlin), and a short walk from the ultra-modern Sony Center and Daimler complex at Potsdamer Platz. Drawing inspiration from the Golden Twenties, the hotel's marbled floors, luxurious polished wood finishes and dazzling chandeliers create a feeling of opulence and supreme comfort. The generously proportioned Hall Maritim, which was home to the Technical stream for the week, was probably the largest room a single stream of the *VB* conference has ever occupied – even making the close to 400 delegates that amassed for the conference opening seem like a relatively small gathering. The Hall Berlin, home to the Corporate stream, was somewhat cosier, but no less elegant.

This was yet another bumper year for delegate numbers – European destinations always seem to draw the crowds and Berlin was no exception, with a grand total of 390 attendees. The growth of the *VB* conference over the years has been fantastic to observe, although the swelling numbers do cause a bit of a headache in that it is becoming increasingly difficult to find venues that have enough space to accommodate us! (A good headache to have, though.)

MAKING A START

VB2013 began on Wednesday morning with the usual opening address and housekeeping notices, after which the stage was left in the capable hands of Andrew Lee, CEO of *ESET* North America. Kicking off the conference in style, Andrew delivered a passionate and provocative keynote address looking at ethics in the anti-malware business in the age of government cyber surveillance. Opening with a video montage on Snowden, the NSA and user privacy, Andrew highlighted the fact that the recent NSA leaks have created a culture of distrust. While the AV industry has spent many years building trust between vendors and among researchers, those levels of trust have been damaged – with researchers no longer sure who can be trusted and who may be disclosing information to intelligence and law enforcement agencies. Kudos to Andrew for managing to slip in quotes from George Orwell and Samuel L Jackson (as Jules Winnfield), some cat pictures and a handful of penguins, while at the same time delivering a thoroughly engaging and thought-provoking talk on a topic that has much relevance to the whole of the security community.

After the keynote address the conference took on its traditional two-stream format. Starting the presentations off in the Corporate stream, Andreas Lindh described ways in which corporations can reduce the window of exposure to zero-day threats by getting back to basics and using tools that are already available as a complement to the often prolonged or delayed patching process.

Next up, Razvan Benchea and Vlad Bordianu described how an examination of over 800,000 apps on the *Google* and *Apple* markets uncovered a worrying number with security issues. The researchers found that 0.44% of *Google Play* apps were using an unencrypted connection for authentication/registration while the same was true for 0.51% of apps in the *Apple App Store*.

Meanwhile, in the Technical stream, James Wyke detailed some of his discoveries about the ZeroAccess botnet, demonstrating its network traffic obfuscation techniques and revenue generating model – and revealing that ZeroAccess click fraud revenue is estimated at between US$90,000 and US$200,000 per day.

After lunch, Tom Cross and Holly Stewart tackled the thorny issue of vulnerability disclosure from a different angle: when

to disclose that a vulnerability is being exploited in the wild. While the knowledge that a vulnerability is being exploited can be helpful for users in prioritizing their defences, the knowledge that a vulnerability exists and that it can be targeted effectively is also useful for potential attackers – and can result in an overall increase in attack activity. Holly and Tom looked at a number of different real-world examples demonstrating both the positive and negative effects of public disclosure of exploitations and showing how the timing of disclosure can significantly affect the outcome.

Later, Joe Blackbird and Bill Pfeifer looked at the global impact of anti-malware protection state on infection rates – they used *Microsoft* statistics to show that PCs without anti-malware protection (or with inadequate anti-malware protection) are 5.6 times more likely to be infected, and a computer that runs consistently without anti-malware has a 10 times greater risk of being infected month on month than a machine with protection installed. Among other things, the pair called for anti-malware vendors to make sure that disabling options for anti-malware products are well hidden.

Wednesday afternoon saw presentations on two major cross-industry telemetry projects. Righard Zwienenberg and Thomas Wegele presented a system being developed by members of the Anti-Malware Testing Standards Organization (AMTSO) that is intended to overcome the shortcomings of the WildList. The Real Time Threat List (RTTL) should offer a more accurate and up-to-the-minute picture of the latest threats in circulation than anything currently on offer, while also providing a flexible framework for testers to make use of it in different ways. Later, Igor Muttik and Mark Kennedy spoke about another cross-industry initiative, this one developed by members of the IEEE Industry Connections Security Group (ICSG) malware research group and designed to help mitigate the issue of false positives. The IEEE-ICSG clean file metadata sharing system (CMX) provides a database that can be shared by all security vendors. Legitimate software developers can submit metadata for their products to the system, and the metadata will be relayed to all security vendors at once, thus enabling vendors to add the products to their whitelists and helping them to build clean sample sets for quality assurance.

The first day drew to a close with sponsor presentations in each stream. In the Corporate stream, *ESET*'s Stephen Cobb asked 'What can Big Data Security learn from the AV industry?', while in the Technical stream *AV-TEST*'s Andreas Marx detailed some of the testing company's expert knowledge and research.

BEER AND WINE MAKES YOU FEEL FINE

Wednesday evening saw the usual gathering of attendees for the VB2013 drinks reception – the wine, beer and canapés

Cheers! VB2013 delegates let their hair down and enjoy some valuable networking.

flowed steadily, giving delegates the chance to unwind and discuss the important issues of the day (such as who would triumph in this year's *G Data* table football tournament, whose beer consumption would top the *Avast* league table, and how on earth did the catering staff manage to superheat the inside of the mini spring rolls to a near thermonuclear degree?).

Indeed, the beer flowed freely throughout the three days of the conference at the temporary bar set up by *Avast* – and an online league table provided live updates as to the most prolific beer drinkers both individually and by company. Wednesday night also saw the lure of all sorts of weird and wonderful alcoholic beverages at *G Data*'s 'Snake Oil' party held after the *VB* drinks reception (one delegate suggesting he may actually have sampled window cleaner disguised as an alcoholic beverage). For those who had been unable to resist doing the drinks party double, Thursday morning brought a scrabbling for aspirin and coffee – and some concerns as to how their livers would stand up to the next assault on the *Avast* beer leader board.

GOING MOBILE AND LAST MINUTE

Returning to the serious stuff, on Thursday morning VB2013 went mobile with a series of presentations on various different aspects of mobile malware threats. Rowland Yu kicked things off with an in-depth look at GinMaster – a piece of *Android* malware distributed via the many third-party app markets in China and which is estimated to bring in approximately US$245,000 per month for the criminals behind it.

Samir Mody tackled the subject of *Android* malware obfuscation – as the volume of *Android* malware grows and more anti-virus vendors provide protection against it, it is likely to be only a matter of time before *Android* malware obfuscation becomes routine, as it has in *Windows* malware. Samir highlighted some of the current methods of obfuscation used in *Android* malware, and showed examples of .dex byte-code and data obfuscation techniques which are likely to be abused in the future.

Axelle Apvrille focused on the security and privacy issues of *Android* ad kits, revealing the shocking level of personal detail collected by most in-app ad kits – including information on age, gender, sexual orientation, marital status, religion, education, income and ethnicity – and concluding that the current mobile ad model is far too heavily weighted in favour of the advertiser.

On a similar theme, Vanja Svajcer looked at the issue of potentially unwanted applications in the mobile environment. The difference between malware, potentially unwanted applications and legitimate apps for mobile platforms is often much less clear than it is within the desktop world – Vanja laid out a set of common criteria that can be used by researchers and developers for detecting and determining the differences between malware and potentially undesirable apps.

Finally in the mobile-themed block, Roman Unuchek looked at the web infections that only lead to malicious redirection if the request comes from a mobile device, detailing some of the major redirection techniques.

Besides mobile-related presentations there were several other highlights on Thursday morning, including Gunter Ollmann describing how penetration testing with live malware has become a must in today's enterprise networks, and the Technical stream saw the start of the last-minute presentations.

The last-minute presentations – the section of the conference set aside for talks that are submitted and selected as close to the conference as possible – kicked off with Gabor Szappanos detailing how targeted attacks hide behind clean applications. Next up, Christy Chung took a look at the facts behind recent South Korean government DDoS attacks, and John Graham-Cumming detailed how open DNS resolvers are used to launch huge DDoS attacks against websites and DNS servers – demonstrating how frighteningly easy it is to launch a massive DDoS attack.

After lunch, the last-minute presentations also visited the topic of *Android* threats, with Adrian Ludwig explaining *Android* security from *Google*'s point of view. He revealed that fewer than an estimated 0.001% of malicious app installations on *Android* are able to evade its multi-layered defences and that, according to the company's data, users are more likely to install non-malicious rooting and SMS fraud apps than traditional types of malware such as spyware, trojans, backdoors and malicious exploits.

Ross Gibb followed with a very popular presentation detailing how he and his colleagues successfully sinkholed around 500,000 bots (roughly half) of the ZeroAccess P2P botnet, working together with ISPs and CERTs worldwide to clean up infections. Next, Robert Lipovsky and Anton Cherepanov looked at the sophisticated and extremely

Adrian Ludwig shares Google's point of view on Android security.

Life is a cabaret! Spectacular performances from German Dance Sensation.

active Hesperbot banking trojan whose activity peaked between July and September 2013.

The final last-minute presentation was given by Dennis Batchelder and Hong Jia of *Microsoft*, who described recent attacks against their and other AV vendors' automation systems via crafted files. They called for the industry to work together to share information about such crafted files and fix systems and processes before this type of attack can cause significant damage.

Two more sponsor presentations rounded off the day on Thursday, with *Avast*'s Peter Kalnai and Jaromir Horejsi asking 'Are *Linux* desktop systems threatened by trojans?' and *Qihoo 360*'s Paul Fan looking in detail at targeted attacks against Chinese online card games.

GIVE 'EM THE OLD RAZZLE DAZZLE

No *VB* conference would be complete without the glitz and glamour of the gala dinner evening – and this year's gala certainly had glitz and glamour in spades. Dance troupe German Dance Sensation opened the evening with a performance full of sequins, bling, feathers and high kicks, and continued to inject plenty of pizzazz into the evening with several revue-style dance numbers that were both colourful and impressively energetic.

The mellow sounds of Oui D'Accord.

A more mellow tone was set for the rest of the evening by musical trio Oui D'Accord who played their own unique blend of musette, tango and jazz. It was a treat to listen to live music performed by a talented group of musicians, and their smooth sounds created the perfect atmosphere for a relaxing end to the evening. (Of course, the end of the dinner wasn't the end of the evening for the dedicated party people in our midst – I hardly need to mention that *Avast*'s beer continued to flow freely and the hotel's bar was packed to the rafters long into the night.)

THE FINAL PUSH

There were a few bleary eyes on Friday morning, but most delegates who came down for the early morning sessions seemed impressively alert (perhaps in comparison with the *VB*-drinks-reception/*Avast*-bar/*G-Data*-party combo the gala dinner night had been a relatively tame one).

Bravely taking the opening slots on Friday morning were Cathal Mullaney, who presented an end-to-end analysis of the Android.Bmaster trojan in the Technical stream, and Eileen Sinnott and Raymond Roberts who looked at new security measures in *AutoCAD* in the Corporate stream. Next up, Fabio Assolini gave an energetic overview of malicious use of PAC (proxy auto-config, or as Fabio and his colleagues have dubbed them 'problem auto-config') files. Attacks using malicious PAC files have reached a level of efficiency whereby an entire bank account can be hacked with just a 1KB file. Fabio showed the evolution of the attacks, how the bad guys are bypassing detection, and some of the messages the attackers have directed at analysts – politely, and not so politely, asking to be left to continue their shady activity uninterrupted.

After a very welcome mid-morning caffeine boost, proceedings continued with Samir Patil taking a close

No, not the latest boy band on the block about to break into song, but a panel of security experts discussing collateral damage in the age of cyberwarfare (L to R: Tom Cross, Gunter Ollmann, Pedram Amini, Mikko Hyppönen and Ryan Naraine).

Thank you to all of the VB2013 speakers (including those not pictured here!).

look at Blackhole spam and how it can be blocked, and Ciprian Oprisa and George Cabau presenting an overview of ransomware. Ciprian and George showed some of the encryption methods used by various types of ransomware and warned that the amount of ransomware in the wild is on the increase.

There were two sleuthing-themed presentations on Friday afternoon: Peter Kruse focused on the investigation of a large phishing cluster operating out of Morocco, while double act Bob Burls and Graham Cluley described how members of the gang behind the SpyEye botnet were tracked down and arrested in the UK and in Estonia.

The final presentation in the Corporate stream was given by Sergey Golovanov, who discussed the reality of the business-to-government malware market and presented details of the activities of two companies: UK-based Gamma International and Italian Hacking Team – which have sold backdoors and spying tools to governments around the world. Sergey also provided one of the most captivating moments of the conference when he screened a recording of his own version of British comedian Tim Minchin's 'Song for Phil Daoust'. Sergey had re-titled the piece 'Song for John Doe' and dedicated it to the unknown creator of malware for law enforcement. Sergey managed to get an entire room of security experts tapping their feet and clapping along to the chorus of the song – a highly entertaining moment, while also being one of the most bizarre (I had to pinch myself to make sure the late night hadn't got the better of me and that I really was witnessing a security expert singalong).

Finally, rounding off the conference in a not too dissimilar vein to that in which it began, a discussion panel tackled the issue of collateral damage in the age of cyber warfare. Led by Ryan Naraine, panel members Tom Cross, Gunter Ollmann, Pedram Amini and Mikko Hyppönen made some strong points on both the definition and the nature of cyber conflict – an important and controversial subject.

UNTIL NEXT TIME...

As is ever the case, this report has barely scratched the surface of what went on over the course of the three days in Berlin. There were many more excellent presentations that I have not been able to mention, and I would like to thank all of the VB2013 speakers, session chairs and panel members for their huge contribution to the event, as well as the conference sponsors (*Avast*, *AV-Test*, *ESET*, *Qihoo 360*, *HP*, *NSS Labs*, *ThreatTrack Security*, *AV-Comparatives*, *Ikarus Software*, *OPSWAT* and *Veszprog*). My thanks also go to the whole of the *VB* team, the onsite crew and the *Cue Media* technicians for their tremendously hard work and the vital role they played in the running of the event.

Thanks to a number of delegates opting to forgo their printed copies of the VB2013 conference proceedings, a donation of £570 has been made to the conservation charity WWF (http://wwf.panda.org/).

Next year sees the conference hop back across the pond to the West coast of the US, with VB2014 taking place in Seattle, WA, from 24 to 26 September. I look forward to seeing you there.

(Photographs courtesy of: Morton Swimmer, Jeannette Jarvis, Andreas Marx, Pavel Baudis and Eddy Willems. More photographs from the event can be viewed at http://www.virusbtn.com/conference/vb2013/ photos and slides from the presentations are available at http://www.virusbtn.com/conference/vb2013/slides/index.)

MALWARE ANALYSIS 1

ANOTHER TUSSLE WITH TUSSIE

Peter Ferrie
Microsoft, USA

When one has a nice idea – such as a tricky method for encoding data – it is common to take that idea and improve on it. It is rare to see someone take such an idea and degenerate it, but that is essentially what we have with the W32/Tussie.B virus.

SECOND PLACE GOES TO...

The virus begins by pushing the host ImageBase value from the Process Environment Block onto the stack. It adds the RVA of the host original entry point to that value, to construct the virtual address of the host entry point. This allows the virus to support applications that opt into Address Space Layout Randomization (ASLR). The virus registers a Structured Exception Handler and then retrieves the base address of kernel32.dll. It does this by walking the InLoadOrderModuleList from the PEB_LDR_DATA structure in the Process Environment Block. The virus assumes that kernel32.dll is the second entry in the list. This is true for *Windows XP* and later, but it is not guaranteed for *Windows 2000* or earlier because, as the name implies, it is the order of loaded modules. If kernel32.dll is not the first DLL that is loaded explicitly, then it won't be the second entry in that list (ntdll.dll is guaranteed to be the first entry in all cases). We have seen the effect of this problem elsewhere recently [1].

IMPORT/EXPORT BUSINESS

The virus resolves the addresses of the API functions that it requires. It uses hashes instead of names, but the hashes are sorted alphabetically according to the strings that they represent. The virus uses a reverse polynomial to calculate the hash. Since the hashes are sorted alphabetically, the export table needs to be parsed only once for all of the APIs. Each API address is placed on the stack for easy access, but because stacks move downwards in memory, the addresses end up in reverse order in memory. The virus does not check that the exports exist, relying instead on the Structured Exception Handler to deal with any problems that occur. Of course, the required APIs should always be present in the kernel, so no errors should occur anyway. The hash table is not terminated explicitly. Instead, the virus checks the low byte of each hash that has been calculated, and exits when a particular value is seen. The assumption is that each hash is unique and thus that when a particular value (which corresponds to the last entry in the list) is seen, the list has ended. While this is true in the case of this virus, it could result in unexpected behaviour if other APIs are added for which the low byte happens to match another entry in the list.

Once the virus has finished resolving the API addresses, it allocates a memory block and writes a random byte to the buffer. We do not know why the write is performed. The virus sets the error mode to prevent warning messages for all of the supported error types, even though none of them will trigger anyway. It registers another Structured Exception Handler and then decompresses an MZ/PE header combination using an offset/value algorithm. The implementation supports writing only to the first 256 bytes of a buffer, but this is sufficient to describe the PE file that the virus uses. This compression format is probably optimal for the purpose – while an RLE format could compress the data further, that gain would be more than outweighed by the size of the decompression code.

JOIN THE DOTS

The virus drops the resulting file – which contains only the headers and a page full of zeroes. The headers are very sparse – they contain almost the minimum number of non-zero bytes that must be set in order for the file to be acceptable. Specifically, the headers contain the minimum number of non-zero bytes for a file that contains a section. For a file that contains no sections, several more bytes could be removed. The dropped file has one section, which is unnamed, to reduce the number of bytes that need to be written during the decompression phase. The section has only the executable flag set. This is an interesting choice, since it does not affect the number of bytes to be decompressed but it does introduce the (infinitely small) risk that a future version of *Windows* will enforce the flag exactly as specified, and thus break the virus. Currently, the setting of the executable flag results in the readable flag being set, even if that is not explicitly the case. The reason for this is to support the mixing of code and read-only data in the same segment, for example in ROM code. However, the CPU does have the ability to mark a segment as only executable, which would result in read-access failures in the case of the virus.

After dropping the file, the virus 'runs' it (despite it containing no code, in a way that is entirely different from the 'virtual code' technique [1]) and specifies that it is the target of a debugger. This action allows the virus to intercept debug events as they occur, which becomes important later. It registers yet another Structured Exception Handler, and then waits for debug events to occur within the child process. The virus is interested only in the breakpoint debug event, and ignores all others. When a breakpoint debug event occurs, the virus checks if it is the first such event. If it is, the event corresponds to the debug breakpoint that is triggered by the thread that *Windows* creates when a process is being

debugged. Since this thread executes before the main thread does, it is the perfect place to perform certain actions before the main thread begins to execute. The virus takes advantage of this to insert some content into the process – in this case a series of call instructions into an array of 'int 3' instructions. The virus then resumes execution of the child process.

BREAK-DANCING

With the new content in place, a series of intentional breakpoint events occurs in the child process, which the virus intercepts. For each of these breakpoint events, the virus registers another Structured Exception Handler, and then queries the execution context for the child process. The virus reads a value from the stack of the child process and uses this to set the pointer to the next instruction to execute within the child process. Note that the stack pointer is not adjusted at this point, leading to stack leakage corresponding to the number of instructions that are executed by the child. Thus, if the child executed a sufficient number of instructions, a stack fault would occur which would trigger an event (technically, two events – a stack overflow exception, followed by an access violation exception) that the virus would ignore, causing the fault to occur again, and leading to an infinite loop.

The virus uses the exception address to decode one byte of code for each iteration, and place it in the virus body. Once the child has finished executing, the virus verifies that the entire body has been decoded correctly. If the decoding is correct then the virus runs the decoded body.

The body begins by searching the current directory (only) for all objects. It is really only interested in files, but it will examine everything that it finds. For each object that is found, the virus will attempt to remove the read-only attribute, open it, and map a view of it. For directories, the open will fail and the map will be empty. For files, the entire file is mapped into memory, if the file can be opened. However, as with many previous viruses by the same author, this one uses only ANSI APIs. The result is that some files cannot be opened because of the characters in their names, and thus cannot be infected. The virus is interested in Portable Executable files for the 32-bit *Intel* x86 platform, which are executable but not DLLs, system files, or ROM images, and which target the GUI subsystem. The virus checks that the optional header size is the standard value (which is a good idea to exclude unconventional modifications such as those made by many runtime compressors). The virus also excludes files that appear to have a Load Configuration Table, because this contains protections such as SafeSEH, which are difficult to modify in a way that would allow the virus to raise exceptions at arbitrary addresses. The virus requires that the file has a base relocation table that begins at exactly the start of the last section, and which is at least as large as the virus body.

TOUCH-AND-GO

If the file passes those checks, the virus marks the last section as writable and executable, and then copies itself over the relocation table. The use of the executable characteristic allows the virus to run in case the file opts into Data Execution Prevention, or if the system enforces it for the process. The virus clears only two flags in the DLLCharacteristics field: IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY and IMAGE_DLLCHARACTERISTICS_NO_SEH. This allows signed files to be altered without triggering an error, and enables Structured Exception Handling. Interestingly, the use of Structured Exception Handling, and thus the need to clear the flag, is essentially optional. The virus could use Vectored Exception Handling instead, which is entirely independent of the value of the flag. The reason for the NO_SEH flag is to increase the security of a process by disallowing the use of attacker-defined exception addresses when a stack buffer vulnerability is exploited. Since Vectored Exception Handlers reside on the heap instead of the stack, they are less vulnerable to buffer overflow exploits.

The virus also zeroes the Base Relocation Table data directory entry. This is probably intended to disable ASLR for the host, but it also serves as the infection marker. Unfortunately for the virus writer, this has no effect at all against ASLR. The 'problem' is that ASLR does not require relocation data for a process to be 'relocated'. If the file specifies that it supports ASLR, by having the IMAGE_DLL_CHARACTERISTICS_DYNAMIC_BASE flag set in the DLLCharacteristics field, and by not having the IMAGE_FILE_RELOCS_STRIPPED flag set in the COFF Characteristics field, then it will always be loaded to a random address. The only difference between the presence and absence of relocation data is that without the relocation data, no content in the process will be altered. *Windows* assumes that if the process specifies that it supports ASLR, then it really does support ASLR, no matter what the structure of the file looks like. The result is that a process that had a relocation table overwritten by the virus will crash when it attempts to access its variables using the original unrelocated addresses. Alternatively, if the platform does not support ASLR (i.e. *Windows XP* and earlier), and if something else is already present at the host load address (or if the load address is intentionally invalid to force the use of the relocation table), then the file will no longer load.

Finally, the virus sets the host entry point to point directly to the virus code and then raises an exception using the 'int 3' technique. The 'int 3' technique appears a number of times in

the virus code, and is an elegant way to reduce the code size, as well as functioning as an effective anti-debugging method. Since the virus has protected itself against errors by installing a Structured Exception Handler, the simulation of an error condition results in the execution of a common block of code to exit a routine. This avoids the need for separate handlers for successful and unsuccessful code completion.

EXCEPTIONAL BEHAVIOUR

The exception handler unmaps the view and closes the file handles, restores the file attributes, and then continues the search for more objects. After all the objects have been examined, the virus raises another exception to unwind further. That exception handler closes the thread and process handles of the debugged process, and then raises yet another exception. At this point, the virus wants to restore the error mode and free the allocated memory, before raising the final exception to run the host code, but there is a bug in this code (technically, two bugs of the same kind): the wrong offset is used when indexing into the structure that holds the API addresses. Instead of calling the SetErrorMode() function, the virus calls the ReadProcessMemory() function, and instead of calling the GlobalFree() function, the virus would call the GlobalAlloc() function if the ReadProcessMemory() function returned successfully. However, the improper parameters on the stack for the ReadProcessMemory() function cause the API to raise its own exception, which the virus intercepts.

The final exception handler restores the registers and transfers control to the host. Of course, since the GlobalFree() function was never called, the virus leaks a small amount of memory as a result. The bug that causes the early exception is the kind of problem that can only be found by single-stepping through the code, since an exception of some kind is the expected behaviour; it just happens too early in this case.

CONCLUSION

The Tussie.A virus [2] showed us a way to hide encoded data by using something approaching the smallest possible implementation of individual opcode decoding. The Tussie.B virus takes a step back by introducing a layer of indirection for no obvious purpose other than to make debugging a bit more difficult. Fortunately, the simplicity of this implementation still results in simplicity of detection.

REFERENCES

[1] http://www.virusbtn.com/virusbulletin/archive/2013/10/vb201310-Eagle.

[2] http://www.virusbtn.com/virusbulletin/archive/2012/08/vb201208-Tussie.

MALWARE ANALYSIS 2

NEUREVT BOT ANALYSIS

Zhongchun Huo
Fortinet, China

Neurevt (also known as Beta Bot) is an HTTP bot [1] which entered the underground market around March 2013 and which is priced relatively cheaply [2]. Though still in its testing phase, the bot already has a lot of functionalities along with an extendable and flexible infrastructure. Upon installation, the bot injects itself into almost all user processes to take over the whole system. Moreover, it utilizes a mechanism that makes use of *Windows* messages and the registry to coordinate those injected codes. The bot communicates with its C&C server through HTTP requests. Different parts of the communication data are encrypted (mostly with RC4) separately. In this article, we will take a detailed look at this bot's infrastructure, communication protocol and encryption schemes. (This analysis is based on samples that were collected from March to June 2013.)

INSTALLATION/DEPLOYMENT

Installation process

Just like most malware, the installation of Neurevt starts with it copying itself to a system folder. The folder is selected according to the machine's characteristics such as the version of *Windows*, the service pack installed, and whether the OS is 64-bit.

For example, on an x86 machine running *Windows XP* SP2, the chosen folder is %PROGRAM FILES%\COMMON FILES. The installer creates a sub-folder named 'winlogon.{2227A280-3AEA-1069-A2DE-08002B30309D}'.

The first part of the folder name, 'winlogon', is obtained from the configuration of the bot, and the second part is a special GUID which makes the folder link to the 'Printers and Faxes' folder in *Windows Explorer*. This folder will act as the launching point each time the malware restarts. The installer then launches the new file and exits.

The newly launched copy creates a process of a system application, which also varies under different circumstances, and starts to inject. The injected data is within a continuous block of memory and has the following data layout:

1. A copy of the whole PE image of the malware process.
2. A block of data which contains states used by the code of the PE image in part 1.
3. A buffer that contains the file data of the malware.
4. Encrypted configuration data.

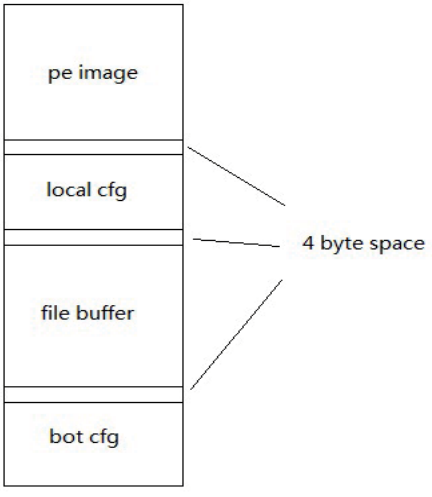

Figure 1: Layout of injected data.

Since this is the first time the malware has injected something into another process, the injected content runs as an independent application. I refer to it as the 'primary instance' to distinguish it from other instances that are injected into other processes.

The primary instance searches for all running processes, and injects into those that fulfil the following conditions:

1. The creator of the process is not services.exe.
2. The user that creates the process is not NT AUTHORITY\SYSTEM.
3. It is not one of the following system processes:

   csrss.exe

   smss.exe

   lsass.exe

   services.exe

   spoolsv.exe

   winlogon.exe
4. It is not created by the current process.

This time, the injected data has the same layout as the primary instance. The only difference is in the 'local cfg' part – in which some data fields are modified to act differently since some components should be loaded in the primary instance only.

After injection, there should be one instance of the malware in every running user process. I refer to these as 'assistant instances'. There is code in every assistant instance that monitors the status of the primary instance. Once, for whatever reason, the primary instance exits, the assistant instance will attempt to restart the malware from its launch point.

After the malware has finished its deployment, the primary instance will start to communicate with the C&C server. The whole process looks like a traditional virus infecting a file system, but instead of infecting files, the malware infects running processes.

Gather local information

The malware performs a system scan to gather information about the system and installed software. The gathered information will be stored in four separate flags, each of which is a single-DWORD-length bit-flag.

The first flag contains information about the *Windows* version, installed service pack, and whether the OS is 32- or 64-bit.

The second flag contains information about the following software or vendors: .Net Framework, Java, *Steam*, *SysInternals* tools, *mIRC*, *Hex-Rays*, *Immunity Inc.*, *CodeBlocks*, *7-Zip*, *PrestoSoft*, *Nmap*, Perl, *Visual Studio* and *Wireshark*. It also contains information that indicates if the system:

1. Has battery
2. Has RDP records
3. Has UAC enabled.

The third flag contains information about the following software or vendors: *Steam*, *EA Origin*, *RuneScape*, *Minecraft*, *Blizzard*, *League of Legends*, *VMware*, *Skype* and *Visual Studio*.

The fourth flag contains information about installed AV software: *Symantec*, *AVP*, *AVG*, *Avira*, *ESET*, *McAfee*, *Trend Micro*, *Avast*, *Microsoft Security Client*, *Bitdefender*, *BullGuard*, *Rising*, *Arcabit*, *Webroot*, *Emsisoft*, *F-Secure*, *Panda*, *PC Tools Internet Security* and *G Data AntiVirus*.

Component thread

The malware creates threads to perform different kinds of jobs, such as communicating with the C&C server, checking data consistency, managing messages passing among threads (components), or monitoring and infecting USB drives. These threads are like software components. In order to load the threads properly, the malware defines a function to create them. The following is the function's definition:

```
HANDLE NewComponentThread(
        LPVOID ThreadProc,
        LPVOID InputBuf,
        int SizeOfInputBuf,
        int idx,
        int reserved,
```

```
    int *pThreadId,
    int flag
    );
```

ThreadProc is the routine that performs a particular job, while idx is the index assigned to it by the malware. 0-0x1E and 0x21 are idx values given to those unique routines for which there should be only one running thread. 0x1F and 0x20 are for multiple instance routines. If the NewComponentThread is called with idx set to these two values, the function will assign a new idx to ThreadProc, which is the first available (unassigned) number from 0x22.

The malware maintains a list to keep track of all the threads that are created by the function. Each ThreadProc takes an entry pointed to by its idx. The entry of the list has the following structure:

```
typedef struct THREAD_LIST_ENTRY {
    WORD Size;
    WORD Index;
    DWORD reserved;
    DWORD CreateFlag;
    DWORD SizeOfInputBuffer;
    DWORD InputBuffer;
    CHAR EventName1[96];
    HANDLE Event1;
    HANDLE Event2;
    HANDLE ThreadHandle;
    DWORD ThreadId;
    DWORD StartAddress;
    DWORD StubAddress;
    DWORD CurrentId;
    };
```

Before actually starting the new 'component thread', NewComponentThread adds a short code stub and a wrapper function to ThreadProc.

The code stub will be written into ntdll's image, at a random location within the MZ header. This random location will serve as the StartAddress when NewComponentThread calls CreateThread. So, the start address of the 'component thread' is within the memory range of ntdll's image. This feature is used when the malware passes messages among its threads.

```
mov     eax, WrapperFunAddr
jmp     eax
```

Figure 2: Code stub.

The wrapper function attempts to hide the thread from debuggers and updates the thread list before and after it calls ThreadProc. It also sets a specific TLS slot with a specific value, 1234. Since the malware's code is always running in an injected process, the API hook will be applied to monitor and manipulate the behaviour of the host process. The specific TLS slot value is used to identify the malware's own threads for which the API hook should not be applied.

API hook

The malware applies the Ring 3 hook in two ways.

First, the malware adds a pre-operation filter for each of the following Zw* APIs:

- ZwCreateFile
- ZwOpenFile
- ZwDeleteFile
- ZwSetInformationFile
- ZwQueryDirectoryFile
- ZwCreateKey
- ZwOpenKey
- ZwSetValueKey
- ZwOpenProcess
- ZwTerminateProcess
- ZwCreateThread
- ZwCreateThreadEx
- ZwResumeThread
- ZwSuspendThread
- ZwSetContextThread
- ZwOpenThread
- ZwUnmapViewOfSection
- ZwDeviceIoControlFile
- ZwQueueApcThread

The filter first checks the specified TLS slot. If its value is 1234, this means that the calling thread belongs to the malware. The filter will do nothing and let the thread call the real API. If the TLS slot is not 1234, the filter examines the object (process, thread, file, registry) on which operation will be performed, if the object belongs to the malware, then the filter will return an error status to the calling thread.

The second way it applies the Ring 3 hook is by applying an inline hook on the following two groups of APIs:

1. getaddrinfo, GetAddrInfoW, DnsQuery_W
2. HttpSendRequestW, PR_Write

The malware hooks APIs in group 1 to block unwanted hosts. The host list is received from the bot server. Most of the unwanted hosts are the web servers of anti-virus software vendors.

The malware hooks the APIs in group 2 only if the injected process is one of the following browser processes:

- firefox.exe
- iexplore.exe
- chrome.

The malware receives a list of URLs to be monitored from the bot server. If the browser sends requests to these URLs, the malware will capture the request data and send it back to the bot server.

Handling messages

There are multiple instances of the malware running in the system. To coordinate these instances, the malware creates a thread in each as a handler of application-defined messages. Most of the messages are sent from the primary instance after it has received something from the bot server to notify other instances to update their local data, such as the blocked host list and the monitored URL list mentioned earlier.

If a malware's thread is about to send a message, it enumerates all the running threads in the system, searching for those that have a start address within ntdll's image. As described in the 'Component thread' section, all the threads created by NewComponentThread fulfil this condition. The sending thread will call PostThreadMessage to send the message to them. Among these threads, only the message handlers (in all the malware instances) have a message queue (by calling IsGuiThread) for messages that are not associated with any window (a feature of messages sent by PostThreadMessage). So the handlers will retrieve the message and response accordingly.

Before a message is sent out, the sending thread will add a pre-defined modifier to the message identifier. This modifier is calculated based on the signature string in the bot configuration and the computer name. Its value is within the range from 0 to 31. The wParam and lParam are also modified since they often carry values with specific meanings like process id or thread id.

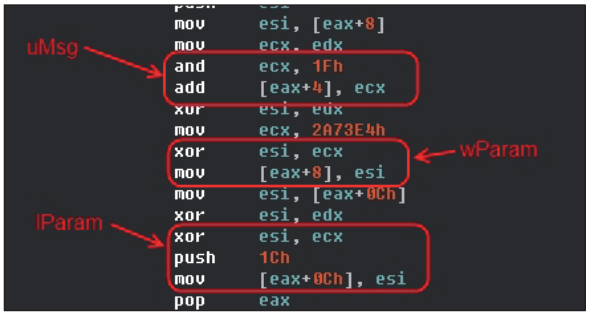

Figure 3: Modify message before sending.

In the handler thread, these values will be restored by a reverse calculation before the message is handled. This means that, even if the messages passing among the malware's threads are being monitored, it's hard to understand their meanings.

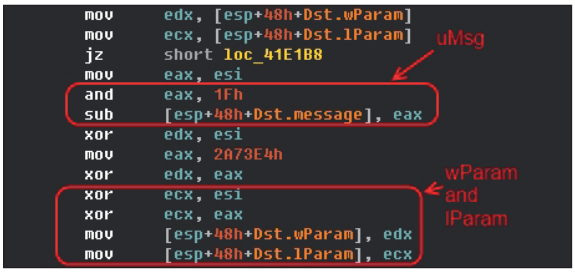

Figure 4: Restore the message before handling.

The messages supported by the handler thread are listed in Table 1:

uMsg	Operation
0xECD	Update the process ID of the primary instance.
0xECB	Update the thread ID of the thread that sends the captured HTTP request back to the bot server, also update the handle of the window created by the thread. The captured data will be sent to the window by the WM_COPYDATA message.
0xEC9	Perform operation (search or insert) on a pid list and a tid list. These two lists are used by hooked Zw* functions. The pids and tids in these two lists belong to the malware and will be protected.
0xEED	Store the Control flag value in the last received bot response.
0xEE9	Update the local blocked host list.
0xEE7	Update the local monitored URL list.
0xEC7	Kill all of the malware's threads in an inject instance.

Table 1: Messages supported by the handler thread.

Sharing data in registry

The malware stores shared data for all instances in registry values. The values will be created under the key HKCU\Software\CLSID\{random guid}\{hash of configuration signature string}. The following are important values used by the malware:

- CS1\S02: Encrypted data that stores the last received blocked host list.
- CS1\S03: Encrypted data that stores the last received monitored URL list.
- CS1\S01: The last received configuration.
- CG1\CF01: Value of the DWORD at offset 0x20 in the last received response.
- CG1\CF02:Value of the DWORD at offset 0x24 in the last received response.
- CG1\CF03: Value of the DWORD at offset 0x28 in the last received response.
- CG1\BIS: Flag that indicates that the process is running on a removable disk.
- CG1\BID: The first launch time.
- CG1\HAL: DWORD, set to 0xEE05 after it has been installed successfully.
- CG1\LCT: Time of last received bot response.

BOT COMMUNICATION

Neurevt communicates with its C&C server through HTTP. Both the request and response are encrypted with the RC4 algorithm. The communication starts as the malware on an infected machine sends its first request to the bot server. Then the communication goes on in a 'Q & A' manner.

Bot configuration

Neurevt has a built-in configuration. The configuration data and the decryption code are both encrypted with RC4. The malware allocates a block of memory on the heap, and decrypts and copies the decryption code into the memory. Then it creates a thread to decrypt the configuration.

The argument passed to the thread is a pointer to the following structure:

```
typedef struct DECRYPT_CONFIG_STRUCT {
   DWORD cbSize;
   DWORD lpConfigData;
   DWORD lpKey;
   DWORD lpGlobalData;
   DWORD lpKeyForReEncrypt;
   DWORD ImageBase;
   DWORD DecryptFunc;
   DWORD SearchFunc; // To search the PE image for
config's hash
   DWORD AllocMemory;
   DWORD ReleaseMemory;
   DWORD HashString; // To calculate the config's hash
   DWORD GetCriticalSection;
   DWORD Win32APIList;
};
```

The fields between DecryptFunc and GetCriticalSection are functions that are used by the decryption function. First, the decryption thread will perform an integrity check by calculating the hash value of the key sequence and comparing it with a value that is embedded in the PE image. If the hash values are consistent, the thread allocates a block of memory and decrypts the configuration data into the memory.

lpKeyForReEncrypt is a pointer to a four-byte key sequence for re-encryption. It is generated randomly after the configuration data has been decrypted. The re-encryption also uses the RC4 algorithm. Its purpose is to protect the configuration data from being discovered in any memory dump. The re-encryption is done by the main thread after the decryption thread exits. Before the decryption thread exits, it stores the memory pointer and the re-encryption key in the memory block that is given by lpGlobalData.

Any access to the configuration data afterwards is carried out using the following steps:

1. Allocate a block of memory and copy the re-encrypted data into it.
2. Decrypt and get the desired data.
3. Re-encrypt the data.
4. Release the memory.

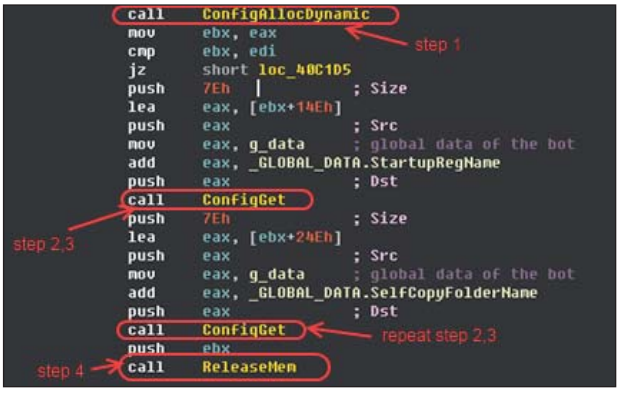

Figure 5: Access configuration data.

Steps 2 and 3 are completed in one function. So, the configuration data remains plain text in the memory for only a short period of time.

The configuration data has a 718-byte header, which contains the fields shown in Table 2.

There is an array at 0x2ce of the configuration data. Each entry stores information about a C&C server. Normally there will be more than three entries in one configuration. The size of the entry is 0x280 bytes. The crucial fields are shown in Table 3.

Offset	Type	Meaning
0x0	WORD	Size of the configuration data, 0x2ace
0x2	DWORD	Magic number
0x6	BYTE[32]	Signature string, used for creating names, such as event, registry
0x44	CHAR[104]	Backup URL
0x14e	WCHAR[128]	Name of the startup registry
0x24e	WCHAR[128]	Name of the folder where the installer self-copies itself

Table 2: Fields contained in the 718-byte header.

Offset	Type	Meaning
0x00	WORD	Entry size, 0x280
0x0e	DWORD	Hash of the domain name
0x12	WORD	Number of attempts
0x14	WORD	Server port
0x16	DWORD	(Length of domain name) xor (0xa5f0 + (index in the array))
0x1E	WORD	Option
0x66	CHAR[256]	Host
0x166	CHAR[256]	Path
0x26e	BYTE[8]	RC4 key 1, for request and response header
0x277	BYTE [8]	RC4 key 2, for response body

Table 3: The crucial fields.

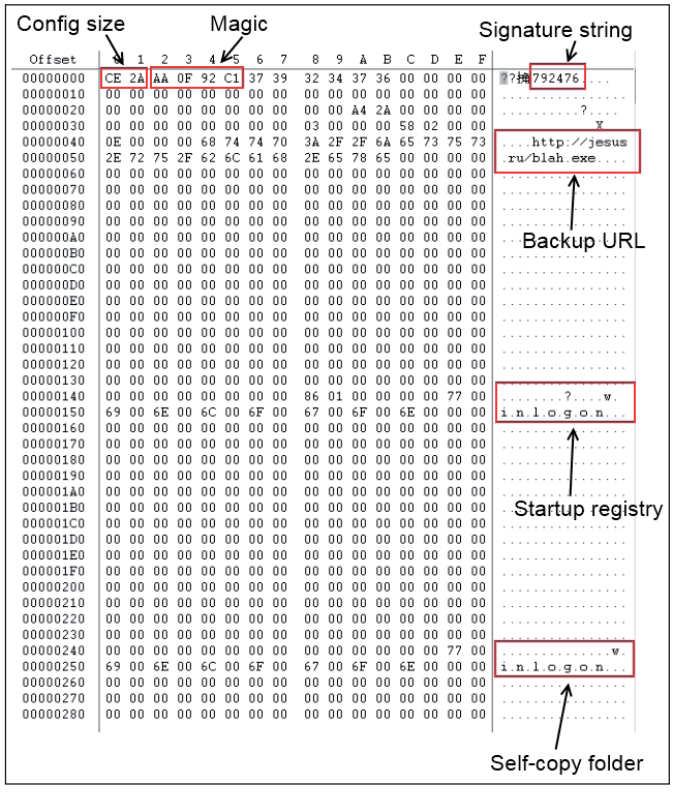

Figure 6: Configuration (header).

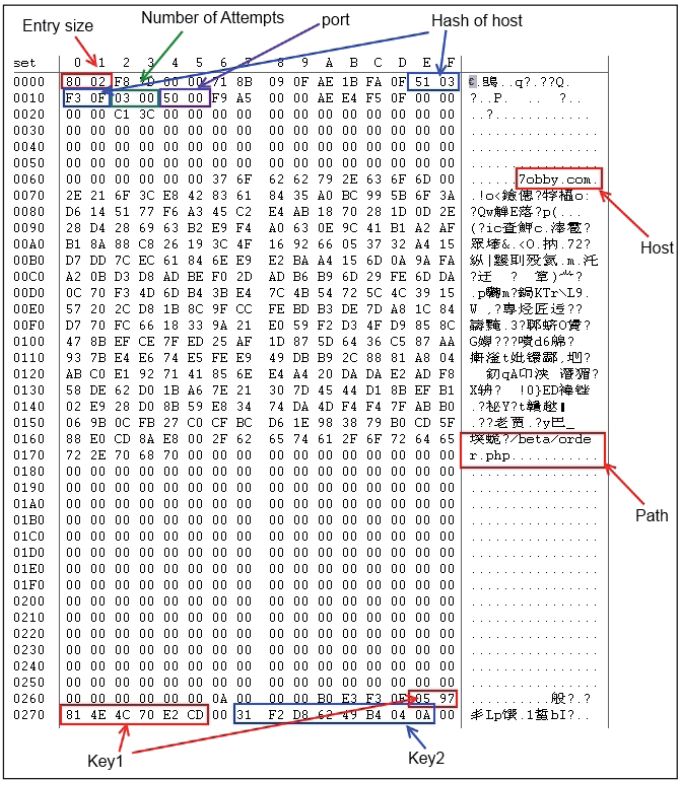

Figure 7: Configuration entry.

Request format

Each request contains a 128-byte data block, which contains the fields shown in Table 4.

The malware encrypts the data block with RC4. It uses a 12-byte key sequence, which is a combination of two parts. The first is an eight-byte sequence obtained from the chosen entry of the configuration data, at offset 0x26e. The second part is a random sequence obtained by calling CryptGenRandom. The length of the sequence is within a range from eight to 27 bytes. This part of the key and encrypted data block will be inserted into the query string.

Offset	Type	Meaning
0x00	BYTE[8]	Not used
0x08	WORD	Size of the block
0x0A	WORD	Magic, 0xC1E5
0x0C	DWORD	Software flag 1
0x10	DWORD	Count of CS fields in the request
0x14	DWORD	Unknown
0x18	DWORD	DWORD read from configuration, at offset 0x30
0x20	DWORD	Software flag 2
0x24	DWORD	Seconds since 1970
0x28	DWORD	Tick count
0x2C	DWORD	TimeZone.Bias
0x30	DWORD	Locale info
0x34	WORD	Data of registry value BK32
0x36	WORD	Reserved
0x38	BYTE[16]	Guid
0x48	DWORD	Data of registry value CF01
0x4C	DWORD	Data of registry value CF02
0x50	DWORD	Data of registry value CF03
0x54	DWORD	Reserved
0x58	DWORD	Security software flag
0x5C	DWORD	Software flag 3
0x60	DWORD	Unknown
0x64	BYTE[28]	Reserved

Table 4: Fields contained in the 128-byte data block.

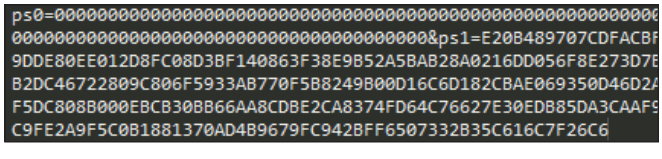

Figure 8: First part of query string.

If it is the first request sent to the bot servers, it will also contain the following three CS fields:

1. A full file path of installed malware
2. The username in NameSamCompatible format
3. The name of the default web browser.

These strings are encrypted separately with a loop-XOR algorithm and concatenated into a URL query string.

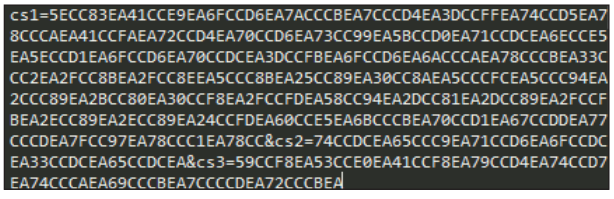

Figure 10: Extra fields for the first request.

Bot response header format

The bot response contains a 0x5c-byte header and a body which consists of at most eight streams (only four streams are actually used). Both the header and the body are encrypted with RC4 and they will be decrypted separately. The first four bytes in the header will be appended to the eight-byte sequence in the configuration to form a 12-byte key sequence for decrypting the header. The following four bytes in the header and another eight-byte sequence in the configuration are concatenated to form another 12-byte key sequence for the response body.

The response header has the structure shown in Table 5.

Offset	Type	Meaning
0x00	BYTE[4]	Key1
0x04	BYTE[4]	Key2
0x08	DWORD	Size of the header, 0x5C
0x0C	DWORD	Response status. 0 if error occurs
0x10	DWORD	Controls the sleeping time during the communication
0x14	BYTE[8]	Reserved
0x1C	DWORD	Control flag
0x20	DWORD	To be saved into registry value CG1\CF01
0x24	DWORD	To be saved into registry value CG1\CF02
0x28	DWORD	To be saved into registry value CG1\CF03
0x2C	BYTE[16]	Reserved
0x3C	DWORD[8]	Length array

Table 5: Structure of response header.

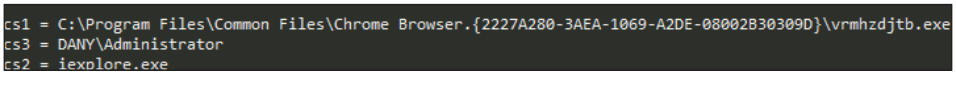

Figure 9: Data of CS fields.

The Control flag in the response header is a bit-flag which triggers different behaviours of the malware, such as invoking routines that disable security software or fake pop-up warning windows to trick the user into giving permission for the malware to bypass the UAC.

Fields from 0x20 to 0x28 are three DWORDs which will be stored in the registry values. They will be copied to the bot server in the next request sent by the malware.

The length array at 0x3C has eight entries – each denotes the length of the corresponding stream in the response body.

Bot response body

According to the response header format, there should be eight streams in the body, but the malware only has handling routines for the first four streams.

The first stream contains bot commands sent back by the bot server. The first word of the stream is the count of the commands in the stream. Each command is stored as a null-terminated string preceded by a data block. The size of the data block is 22 bytes. It is not used in any of the malware's code.

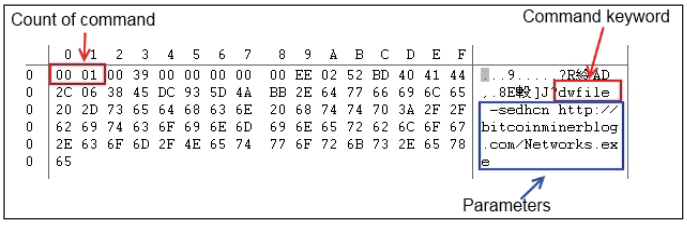

Figure 11: First stream (commands).

The malware identifies the command keyword by hash value. It has a list of handling routines. Each entry in the list has the following structure:

```
typedef BotCmdEntry {
        DWORD dwHash;
        PVOID lpfnCmdProc;
        DWORD KeywordLength;
}
```

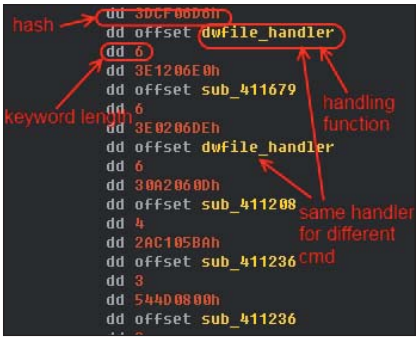

Figure 12: Handling routine list.

The second stream contains a list of hosts that will be blocked. The list is used in the following hooked functions:

- DnsQuery_W
- GetAddrInfoW
- getaddrinfo

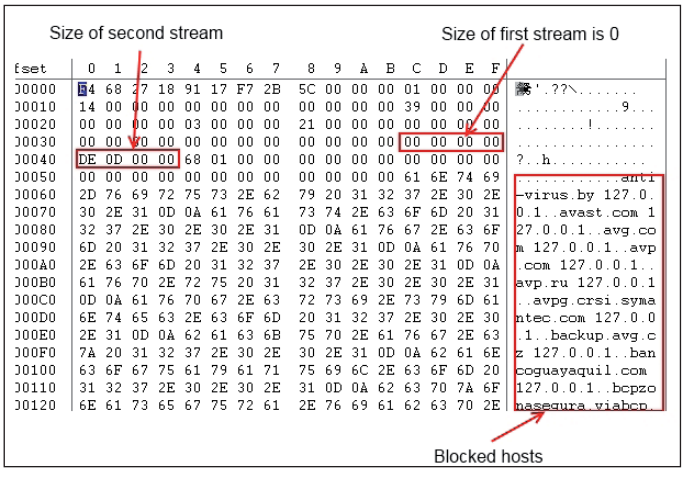

Figure 13: Second entry (blocked hosts).

The third stream contains a list of URLs for which the malware will monitor the HTTP requests sent. The list will be used in the following hooked APIs:

- HttpSendRequestW
- PR_Write

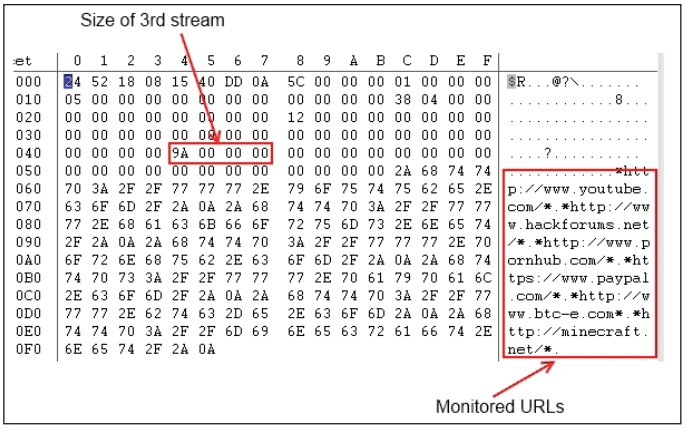

Figure 14: Third entry (monitored URLs).

The fourth stream in the response body contains data which could be used to generate a new configuration. The stream is in a format similar to that of an INI file. The malware compiles the stream into a binary data block which is organized in a structure described in the configuration section.

```
[BackupUrl=http://jesus.ru/blah.exe]
[Server=0]
Host=strike-file-hosting.us
Port=80
Path=/b/order.php
Key1=5449209474046789
Key2=9959708672279946
NumberOfAttempts=3

[Server=1]
....
```

Figure 15: Fourth entry (new configuration).

SPAM THROUGH SKYPE

The malware uses *Skype* to spread any text material received from the bot server. There are two bot commands that will invoke the spreading job.

```
dd 30A9060Ch
dd offset SkypeOperation
dd 4
dd 6EE4094Dh
dd offset SkypeOperation
dd 0Ch
```

Figure 16: Bot commands involving Skype.

The bot server sends the command along with a URL parameter pointing to a text file. Each line of the text file contains a locale-message pair which is delimited by a semicolon:

```
{locale name};{spam content}
```

The malware chooses one line according to the locale of the system's default language and sends the message in the line to all the *Skype* contacts except 'echo123', which is the name of *Skype*'s echo service.

To send the message, the malware creates a new process of itself with the command line parameters set to '/ssp {URL sent by bot server}'. The new process sets up a communication between itself and the *Skype* client with the *Skype* API. Then it starts to send *Skype* commands.

The first command sent is 'SEARCH FRIENDS', which retrieves all the contacts of the logged-in user. For each contact, a 'MESSAGE' command will be sent to the *Skype* client to generate an IM message to send the chosen spam content.

BYPASSING UAC

On a UAC-enabled system, if the malware needs to elevate its privilege, it doesn't play some trick to avoid prompting the user or disable the UAC once and for all. Instead, it will directly 'ask' the user for approval.

The malware creates a process of 'Cmd.exe' and puts the malware's file path in the command line argument. When the prompt window pops up, it shows that 'Cmd.exe', which is a *Windows* application, is asking for privilege elevation. Careless users tend to approve the request. Then a new process of the malware will be created by 'Cmd.exe' and it will inherit the system privileges of 'Cmd.exe'.

Figure 17: Posing as 'Cmd.exe'.

Clicking 'show detail' reveals the trick (Figure 18).

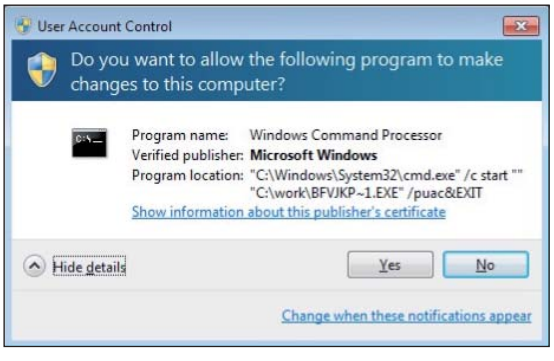

Figure 18: Hidden in detail information.

Under some circumstances, the malware will give a fake warning about system corruption and ask the user to approve a 'restoration utility' to gain a high privilege (Figure 19).

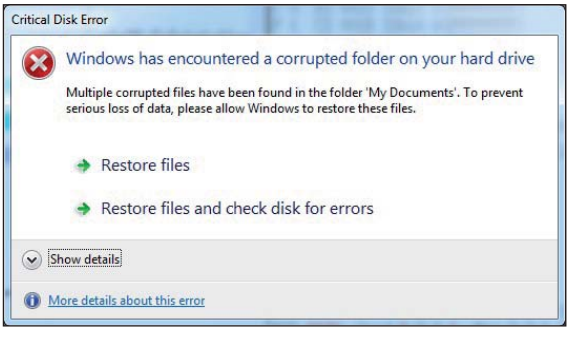

Figure 19: Fake warning (1).

If the user denies it, the malware will continue to pop up warnings and re-prompt the user several times.

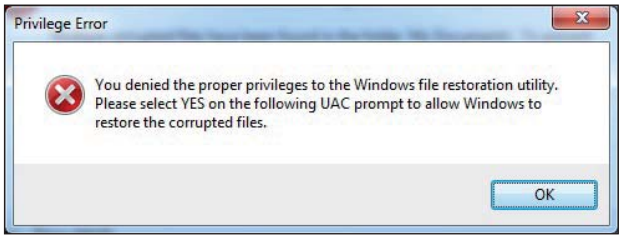

Figure 20: Fake warning (2).

The malware prepares warnings in the following languages: Russian, Portuguese, German, French, Dutch, Arabic, Hindi, Persian, simplified Chinese, Turkish, Indonesian, Italian, Spanish, Polish, Japanese and Vietnamese.

It will choose one of them according to the system's default language locale to avoid language inconsistency raising the user's suspicion.

```
                        ; CODE XREF: sub_426442+26D↑j
cmp     ax, 1Fh
jnz     short loc_4266D2
mov     eax, offset aWindowsDosyaRe ; "Windows dosya restorasyon program"
mov     ecx, offset aIzinHatas ; "Izin hatas"
jmp     short loc_42674F
-----------------------------------------------------------

                        ; CODE XREF: sub_426442+282↑j
cmp     ax, 21h
jnz     short loc_4266E4
mov     eax, offset aAndaMenyangkal ; "Anda menyangkal hak-hak istimewa yang t"...
mov     ecx, offset aPrivilegeKesal ; "Privilege Kesalahan"
jmp     short loc_42674F
-----------------------------------------------------------

                        ; CODE XREF: sub_426442+294↑j
cmp     ax, 10h
jnz     short loc_4266F6
mov     eax, offset aHaiNegatoIPriv ; "Hai negato i privilegi necessari a Wind"...
mov     ecx, offset aErroreNeiPrivi ; "Errore nei privilegi"
jmp     short loc_42674F
```

Figure 21: Fake messages in different languages.

CONCLUSION

As this analysis shows, the communication protocol has reserved enough space for new functionalities to be added in the future. There are unused fields in both the request and response structure and there are four streams in the response data that don't have any code to handle. Though Neurevt has just entered the market, the bot's author is likely to develop it rapidly – we will undoubtedly see new features of Neurevt soon.

REFERENCES

[1] Beta Bot – Coded in C++ – Incredibly Advanced HTTP Bot. http://www.sinister.ly/showthread.php?tid=5234.

[2] A new bot on the market: Beta Bot. http://blog.gdatasoftware.com/blog/article/a-new-bot-on-the-market-beta-bot.html.

MALWARE ANALYSIS 3

WHEN ZACCESS BECOMES A DEBUGGER

He Xu
Fortinet, Canada

ZAccess (a.k.a. ZeroAccess) is a complex and infamous botnet. Since 2009, we have observed many different variants and significant updates. In June 2013, we found and analysed some variants which integrated a debugger engine. This article takes a look at some of the features in those variants.

UNIFIED PACKER

The new variants come with a unified packer that can support both EXE and DLL file formats. The packer uses its original characteristics in the PE header to replace the embedded malware. In other words, the embedded file characteristics will be decided by the packer's value.

```
; int __stdcall start(HMODULE hLibModule, int, int)
                public start
start           proc near               ; DATA XREF: HEADER:Optional

hLibModule      = dword ptr  4
arg_4           = dword ptr  8

                mov     ecx, ds:e_lfanew
                mov     eax, 2000h
                test    ds:(FileHdr_Characteristics - 0F0h)[ecx], ax
                jnz     short RunAs_DLL

RunAs_EXE:
                call    RunAs_Debugger_Debuggee
; ---------------------------------------------------------------------------

RunAs_DLL:                              ; CODE XREF: start+12↑j
                mov     eax, [esp+arg_4]
                dec     eax
                jnz     short loc_401DF3
                push    esi
                push    [esp+4+hLibModule] ; hLibModule
                call    ds:DisableThreadLibraryCalls
                xor     esi, esi
                push    esi             ; lpThreadId
                push    CREATE_SUSPENDED ; dwCreationFlags
                push    esi             ; lpParameter
                push    offset Thread01 ; lpStartAddress
                push    esi             ; dwStackSize
                push    esi             ; lpThreadAttributes
                call    ds:CreateThread
                mov     hThread01, eax
                cmp     eax, esi
                jz      short loc_401DF2
                push    esi
                push    eax
                call    ds:ZwResumeThread

loc_401DF2:                             ; CODE XREF: start+45↑j
                pop     esi

loc_401DF3:                             ; CODE XREF: start+1E↑j
                mov     al, 1
                retn    0Ch
start           endp
```

Figure 1: Code to run as EXE or DLL.

```
RunAs_Debugger_Debuggee proc near          ; CODE XREF
                push    0
                push    4
                push    offset DD_ProcWow64
                push    1Ah
                push    0FFFFFFFFh
                call    ds:ZwQueryInformationProces
                call    ds:IsDebuggerPresent
                test    eax, eax
                jz      short loc_401D83
                call    RunAs_Debuggee
                jmp     short ExitProc
; ---------------------------------------------------

loc_401D83:                                ; CODE XREF
                mov     eax, large fs:18h
                mov     eax, [eax+30h]
                mov     eax, [eax+10h]
                push    dword ptr [eax+44h] ; pCmdL
                push    dword ptr [eax+3Ch] ; lpApp
                call    RunAs_Debugger

ExitProc:                                  ; CODE XREF
                push    0                  ; uExitCode
                call    ds:ExitProcess
```

Figure 2: To run EXE main routine.

The packer will overwrite the embedded sample's PE checksum value. In most cases, the value is 0x313 or 0x200 – which might be the packer version.

When the malware has been unpacked, the packer hands control to the loader component.

LOADER

The loader checks the PE header's characteristics to determine whether it should be run as EXE or DLL (Figure 1). If the marker is set as DLL, the loader simply creates a new thread (which we will talk about later). If the marker is set as EXE, it will check the debugger status and run the file either as a debugger or as a debuggee (Figure 2).

DEBUGGER

In the first instance, the malware will load as a debugger. After that, it will create itself as a debuggee with the flags: DEBUG_PROCESS | CREATE_PRESERVE_CODE_AUTHZ_LEVEL. Next, the debugger instance will start a new thread to create and try to read the mailslot named '\device\mailslot\uewuyew<PID>'.

When all initialization tasks have been completed, the debugger process will loop to wait for further debugging events. The debugger supports the following debug event types:

```
EXCEPTION_DEBUG_EVENT
CREATE_THREAD_DEBUG_EVENT
CREATE_PROCESS_DEBUG_EVENT
EXIT_THREAD_DEBUG_EVENT
EXIT_PROCESS_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT
UNLOAD_DLL_DEBUG_EVENT
OUTPUT_DEBUG_STRING_EVENT
```

DEBUGGER TRICKS

There are some tricks in the loop routine:

1. The debugger will not wait for the debuggee. If there is no debug event, or there is no debug event in the local pending pool, the debugger will jump out of the loop and exit directly, which will cause the debuggee to terminate passively.
2. The debugger will examine the dwFirstChance value of every exception event. It will kill the debuggee if the value is zero. For this condition, *WinDBG* will always reset the value when the breakpoint is triggered. As a result, it is a very effective anti-debug method.

DEBUGGEE

The debuggee code is very simple. It will try to load the system module untfs.dll, and get one API address by ordinal 2302h (Figure 3) – which does not exist in the system

```
push    offset aUntfs_dll ; lpLibFileName
call    ds:LoadLibraryW
test    eax, eax
jz      short retn
lea     ecx, [ebp+var_4]
push    ecx
push    2302h
push    0
push    eax
call    ds:LdrGetProcedureAddress
test    eax, eax
jl      short retn
mov     eax, ds:dword_40003C
mov     eax, ds:(OptionalHeader_CheckSum - 0F8h)[eax]
test    eax, eax
jnz     short loc_401D2F
mov     eax, '3CPS'
push    offset __ImageBase ; "MZ"
push    eax
push    hThread01
call    [ebp+var_4]
```

Figure 3: Load DLL and API by ordinal 2302h then call it.

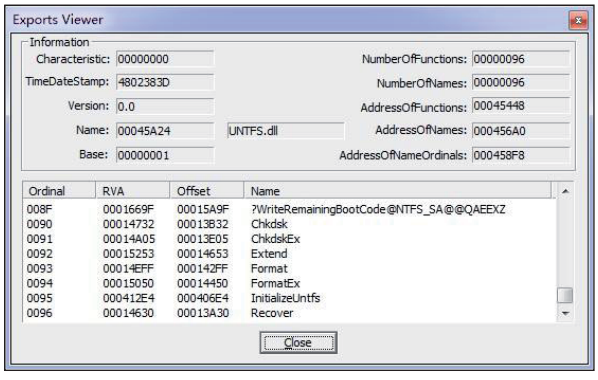

Figure 4: The largest ordinal is 0x9A in system module untfs.dll in Windows XP.

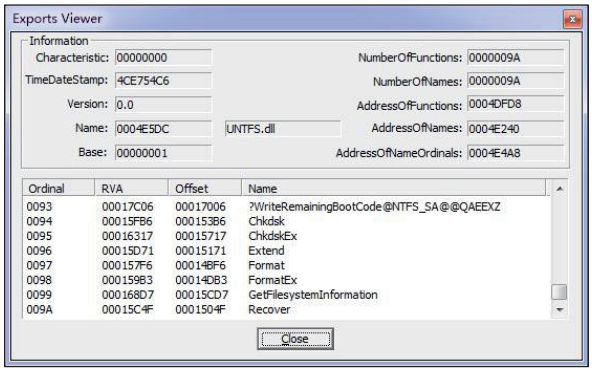

Figure 5: The largest ordinal is 0x9A in system module untfs.dll in Windows 7.

```
lea     eax, [ebp+Context]
push    eax
push    [ebp+arg_hThread]
mov     [ebp+Context.ContextFlags], 10001h
call    ds:ZwGetContextThread
test    eax, eax
jl      short loc_40136C
or      [ebp+Context.EFlags], 100h
lea     eax, [ebp+Context]
push    eax
push    [ebp+arg_hThread]
call    ds:ZwSetContextThread
```

Figure 6: Set the single step marker in the EFlags register.

module (see Figures 4 and 5). You would think, therefore, that this operation would fail.

However, that is not the case here: the debugger will not let the debuggee fail over this non-existent ordinal. So what does the debugger do?

The debugger replaces the newly loaded system module in the debug event processing routine.

SET EFLAGS SINGLE STEP MARKER

First, the debugger will set the single step marker in the EFlags register when processing the DEBUG_EVENT ID 06, as shown in Figure 6.

When the debuggee runs, the debugger will receive the SINGLE_STEP event at the next DEBUG_EVENT with ID 01, and will trigger an exception (Figure 7).

EXTRACT MALICIOUS DLL

The debugger will extract the final malicious DLL from a customized structure with the special signature AP32 (see Figure 8).

In fact, the structure is from aPLib source code and the detail is as below:

```
;   offs size   data
;   --------------------------------
;   0  dword    tag ('AP32')
;   4  dword    header_size (24 bytes)
;   8  dword    packed_size
;   12 dword    packed_crc
;   16 dword    orig_size
;   20 dword    orig_crc
```

At last, we can get the DLL without entry point code (see Figure 9), but including only one export function.

According to the export table, the function has the ordinal 2302 (see Figure 10), but without a name. This is

```
mov     eax, [esi+DbgEventException.EXCEPTION_DEBUG_INFO.ExceptionRecord.ExceptionCode]
cmp     eax, DBG_CONTROL_C
jz      find_Dbg_Ctrl_C
cmp     eax, EXCEPTION_BREAKPOINT
jz      find_BreakPoint
cmp     eax, EXCEPTION_SINGLE_STEP
jnz     Opt_Other
lea     esi, [edi+Heap30h.field_20]
cmp     [esi], ebx
jz      Opt_Other
cmp     [ebp+var_4], ebx
jz      short loc_401A34
lea     eax, [ebp+var_8]
push    eax
call    Dec_DLL         ;
```

Figure 7: Processing exception types.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00001AD0   41 50 33 32 18 00 00 00 B1 9C 01 00 68 99 C6 B0  AP32....±œ..h™Æ°
00001AE0   00 B6 01 00 03 AB AB BC 4D 38 5A 90 38 03 66 02  .¶...««¼M8Z.8.f.
00001AF0   04 09 71 FF 81 B8 C2 91 01 40 C2 15 C6 F0 09 1C  ..qÿ.¸Â‘.@Â.Æð..
00001B00   0E 1F BA F8 00 B4 09 CD 21 B8 01 4C C0 0A 54 68  ..°ø.´.Í!¸.LÀ.Th
00001B10   69 73 20 0E 70 72 6F 67 67 61 6D 87 63 47 6E 1F  is .proggam‡cGn.
00001B20   4F 74 E7 62 65 AF CF 75 5F 98 69 06 44 4F 7E 53  Otçbe¯Ïu_˜i.DO~S
00001B30   03 6D 6F 64 65 2E 0D 89 0A 24 4C 44 7A 01 EA 5C  .mode..‰.$LDz.ê\
```

Figure 8: Malicious DLL under aPLib structure.

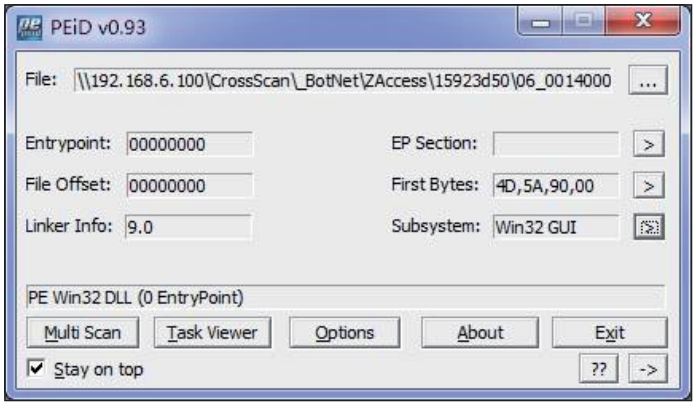

Figure 9: Malicious DLL without EntryPoint.

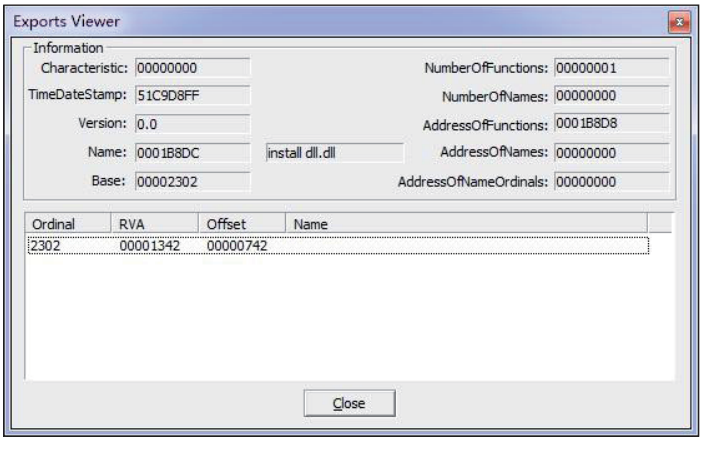

Figure 10: Malicious DLL export table.

exactly what the debuggee wants. The debuggee can then enter the real malicious DLL export function as shown in Figure 3.

THE ZACCESS DLL FEATURE

The DLL code is similar to that seen in previous variants [1]. It also embeds an *MS Cabinet* file (see Figure 11).

Within the code, we can easily see the names of the different modules, such as s32, s64, n64, n32 and fp.exe. The malware connects to j.maxmind.com to test for an Internet connection. If the connection is successful, it will execute the main botnet routine as usual.

CONCLUSION

The use of the debugger/debuggee trick makes the malware much more difficult to analyse in dynamic mode, demonstrating that the ZAccess author is a pretty proficient programmer. The combination of debugger code and botnet features could cause much confusion in distinguishing malicious from clean code.

REFERENCE

[1] Tan, N.; Yang, K. ZAccess detailed analysis. Virus Bulletin, August 2012, p.4. http://www.virusbtn.com/virusbulletin/archive/2012/08/vb201208-ZAccess.

```
4D 53 43 46 00 00 00 00 A7 3D 01 00 00 00 00 00  MSCF....§=......
2C 00 00 00 00 00 00 00 03 01 01 00 05 00 00 00  ,...............
4D 42 00 00 93 00 00 00 05 00 03 15 00 08 00 00  MB..“...........
00 00 00 00 00 00 58 41 9A 7B 20 00 73 33 32 00  ......XAš{ .s32.
00 08 00 00 00 08 00 00 00 00 58 41 B3 7B 20 00  ..........XA³{ .
73 36 34 00 00 66 00 00 00 10 00 00 00 00 7B 42  s64..f........{B
15 77 20 00 6E 36 34 00 00 90 00 00 00 76 00 00  .w .n64......v..
00 00 D9 42 A2 8E 20 00 6E 33 32 00 A0 5C 01 00  ..ÙB¢Ž .n32. \..
00 06 01 00 00 00 37 3F 73 84 20 00 66 70 2E 65  ......7?s„ .fp.e
78 65 00 EF 8F 58 D3 D8 3E 00 80 5B 80 80 8D 08  xe.i.XÓØ>.€[€€..
```

Figure 11: An MS Cabinet file is embedded.

FEATURE

THE MURKY WATERS OF THE INTERNET: ANATOMY OF MALVERTISING AND OTHER E-THREATS

Bianca Stanescu, Ionut Radu & Cornel Radu
Bitdefender, Romania

Unfortunately, reputable companies are not the only entities that use advertising platforms. Scammers are doing their best to tap into more and more of the commercial market. Almost 10 billion ad impressions were compromised by malvertising in 2012, according to the Online Trust Alliance (OTA) [1].

Millions of users worldwide are exposed to malware, spam, phishing or fraud (scams), and even the most tech-savvy users can become victims. Over 100 advertising networks are serving compromised display advertising, and malvertising incidents increased by more than 250% from Q1 2010 to Q2 2010, the OTA showed. At the same time, it is estimated that more than one million sites carry advertising from over 300 ad networks and exchanges, according to the IAB [2]. This means that one in three ad networks may be serving malvertising.

A *Bitdefender* team decided to investigate the anatomy of malvertising and other e-threats (fraud, spam and phishing) injected into legitimate ad networks.

We first focused on the internal structure and web categories of the baits injected by scammers into legitimate advertisements. Our observations suggest that business and computer/software-related landing pages are more lucrative than pornographic ones.

We also recorded the countries in which the fraudulent domains were registered. Some registrants were native to the countries in which the domains were registered, while others were just displacements used by cybercriminals to avoid detection and law enforcement.

After looking at the evolving phenomenon of malvertising, we offer some guidelines to help users and legitimate advertisers avoid these threats. By knowing more about the anatomy of malvertising, companies, security experts and users will be better equipped to fight these emerging security threats.

MALVERTISING AND FRIENDS: THE DISSECTION

Malicious advertising, or malvertising, allows cybercriminals to spread malicious files through legitimate web pages. In September 2009, *New York Times* readers were redirected to a site hosting malware because of an injected ad. One year later, *TweetMeme* (which closed its doors in 2012) suffered a scareware attack because of malvertising. These examples show that malvertising has become a dangerous threat, as it can easily spread across a large number of legitimate websites without compromising the sites directly. Moreover, silent malvertising allows scammers to infect users without the need for any clicking or direct interaction.

According to the OTA, cybercriminals have two main methods to exploit advertising [2]:

'An increasing trend has been to create a fictitious identity and "front" purporting to be a legitimate advertiser or advertising agency. They provide upfront payment and often approach unsuspecting partners with the urgency of a breaking ad campaign. They simply provide the ad creative which appear[s] legitimate on the surface.'

The second and more 'traditional' approach is to breach a vulnerable server to obtain login credentials and then compromise legitimate ads to stay undetected. According to our research, the following are the most common methods used by cybercriminals to spread malicious code through advertisements:

- Pop-up ads for fictitious downloads (e.g. fake movie players, toolbars, plug-ins and media converters)
- Hidden and obfuscated JavaScript code
- Malicious banners
- Third-party advertisements through sub-let ad networks and content delivery networks
- Use of iframes to embed malware and to avoid detection.

Cybercriminals take advantage of two key features of Internet advertising:

- **Dynamism:** online advertising is a versatile medium that allows scammers to stay undetected, as web page content changes regularly. This open system relies on multiple parties, including advertisers, ad networks, ad exchanges, ad services and site publishers, so cybercriminals can easily obscure their trail.
- **Externalization:** companies pay third-party ad networks to distribute ads on their websites without knowing their content and purpose. This allows cybercriminals to pose as legitimate advertising clients. Some fraudulent commercials also appear because big ad networks sub-let some 'advertorial' space to third parties, usually smaller platforms. In this process, the smaller networks may end up placing malicious ads on reputable websites.

Our study of malvertising and other e-threats ran between 23 July and 24 August 2013, when the research team randomly selected over 70,000 ads served on nearly 150,000 websites on the greyer area of the Internet. To select the ads, we scanned search engines for over 50 relevant search terms such as 'download cracks', 'lose weight now', 'earn money at home', 'free movies', 'free music', 'games' and 'torrents'. Because ads change regularly, we also re-tested the web pages for new commercials to increase our database.

In total, 41,400 advertisements led to the same number of landing pages with non-identical URLs. Some were composed of the same domain and path, but with different parameters. These parameters help ad networks retrieve information about user behaviour, such as the website on which users initially clicked on the ad.

To analyse the malware, fraud, spam and phishing prevalence, we put the domains under the magnifying glass. We discovered that only 15,037 unique domains were hosting the 41,400 landing pages, which means that malvertisers and 'friends' place the same baits on different websites for increased efficiency.

We designed a script to open the URLs with a browser emulator that simulated user behaviour by clicking on the ads and retrieving the landing pages automatically. We also retained the redirection chain, as most ads redirect users to two or three other websites, which register them as visitors to add extra revenue for the publishers. According to our data, the malicious ads usually have more redirects than legitimate or 'neutral' ads[1].

We analysed the initial page (client page), the first URLs where the ads were redirecting to, and the final landing page. After we had concatenated the landing pages and applied an MD5 algorithm, we obtained a unique list of signatures. We scanned them with the *Bitdefender* engines to check if any were blacklisted. Almost 7% of the landing pages analysed were blacklisted. The 'neutral' ads represented 46%, one percentage point less than the legitimate ones.

After checking the reasons for blacklisting, we discovered that the majority of the dangerous landing pages were fraudulent URLs luring readers with fake software, business and financial offers (57.54%). After this came spam (14.89%) and malware (14.52%), followed by phishing. With only 4% prevalence, phishing is spread less through advertising networks because users are becoming increasingly aware of such attacks.

[1] 'Neutral' ads were classified as such because there wasn't sufficient data to classify them otherwise, as the sites they led to only had a few visitors and no user ratings.

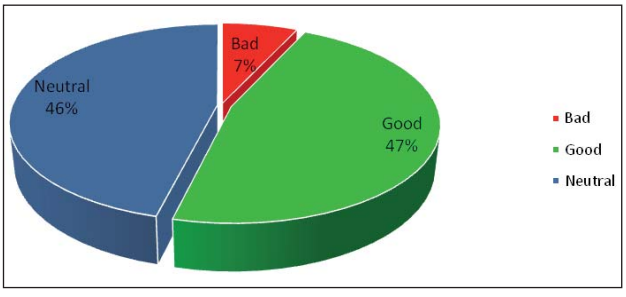

Figure 1: Distribution of good, bad and neutral ads.

An analysis of malicious *Facebook* domains in 2012 [3] also showed that phishing is less distributed on the social network. The research on over 20,000 domains revealed that cybercriminals prefer more effective weapons, such as malware (54%) and fraud (34%), followed by spam (11%) and phishing (1%).

Some of the landing pages we analysed belonged to several categories. The most common combination is phishing with a 'sense' of malware. In this way, if the attackers don't get users' money and personal data through the phishing attack, they install malicious files on the system for similar or worse repercussions. *Bitdefender* also classified close to 9% of the analysed web pages as 'untrusted'.

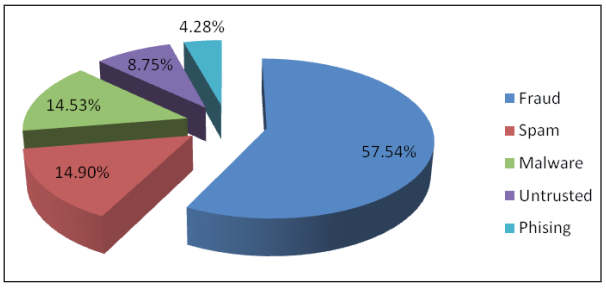

Figure 2: Different types of advertising security threats and their distribution.

INTERNAL STRUCTURE: CATEGORIES

To determine the internal structure of malvertising and other e-threats, and to distinguish the relationships between its components, we determined the web categories of the dangerous landing pages. Our research shows that malvertisers make more money from computer and software, business and health categories, than from pornography.

Similar research this year also showed that malware is more likely to be spread via online advertising than via porn. According to *Cisco*'s 2013 Annual Security Report [4], online advertisements are 182 times more likely to

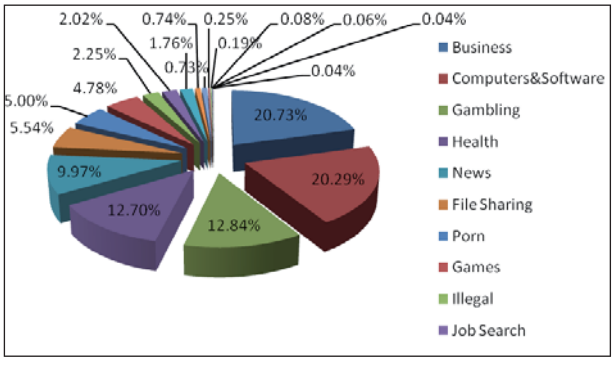

Figure 3: Most dangerous web categories promoted via malvertising and other e-threats.

infect users with malware than searching the Internet for adult content. The report also revealed that the highest concentration of online security threats is found not among pornographic, pharmaceutical, or gambling sites, but on major search engines, retail pages and social networks [5].

The malvertising landing pages were assigned to one of 19 categories, based either on the category of the domain or on the content of that specific web page. For instance, a personal blog hosted on a blogging platform will be categorized as a blog, but it could also be categorized as a gambling page, based on its content.

In the following sections we describe the most lucrative web categories promoted through malware, spam, fraud and phishing placed on legitimate ad networks.

1. Business: 20.73%

The dominant malvertising category covers websites that promote private businesses (corporate websites). Fraudsters create sites that pose as legitimate businesses and target users with fake offers, usually at very low prices.

Unlike phishing sites created by breaching vulnerable websites or domains, fraudulent sites are often well crafted and have web pages registered for longer periods of time. Because they can easily be mistaken for authentic companies, it takes longer before they are taken down, so their uptime is higher than that of phishing sites [6].

Examples of malvertising fraud include fake offers for online garage sales, web hosting or satellite services (Figure 4).

2. Computers and software: 20.29%

This category covers websites that provide computer-related information, software or Internet-related services.

Malvertising observed in this category included a fake *Disney* website that promoted sexually explicit cartoons. Other dangerous ads led users to fake downloads for SEO plug-ins, video convertors and cursors.

3. Gambling: 12.84%

This category covers online casino or lottery websites, which usually require a transfer of funds before the user can start to gamble. Dangerous landing pages found on ad networks trick users into sending their money and personal data with no chance of winning anything. 'Gambling' web pages also include 'beating tips and cheats' websites, which describe how to make money this way.

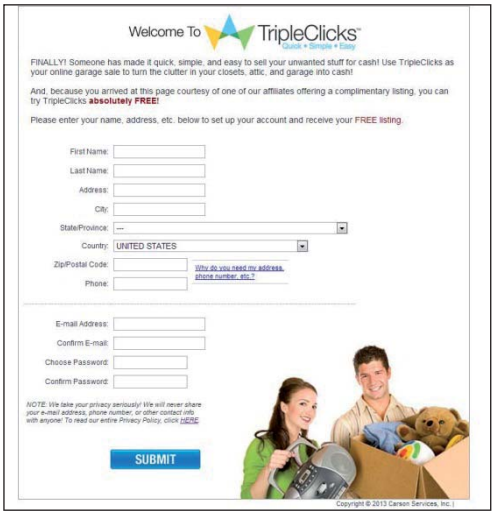

Figure 4: Fraudsters create sites that pose as legitimate businesses and target users with fake offers.

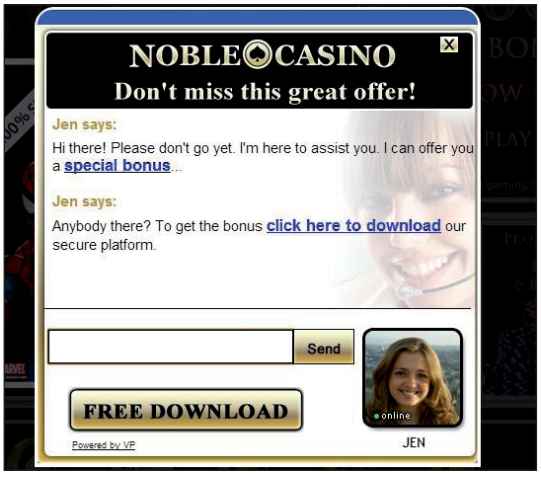

Figure 5: Gambling sites.

4. Health: 12.7%

The fourth most popular malvertising category typically covers websites associated with medical institutions, disease prevention and treatment, and websites that promote weight loss and pharmaceutical products, diets, etc. Malicious ads in this category offer miraculous secrets for 'a tiny belly', fraudulent detox medication and weight loss advice presented in the form of news (Figure 6).

5. News: 9.97%

This category covers news websites that provide journalistic text, video content or newsletter services (Figure 7). It includes both global and local news websites. Our research showed that one in 10 dangerous landing pages were offering content presented as news. Typical fraudvertising cases include fake newsletter offers and weight-loss tips presented by a medical news 'reporter'.

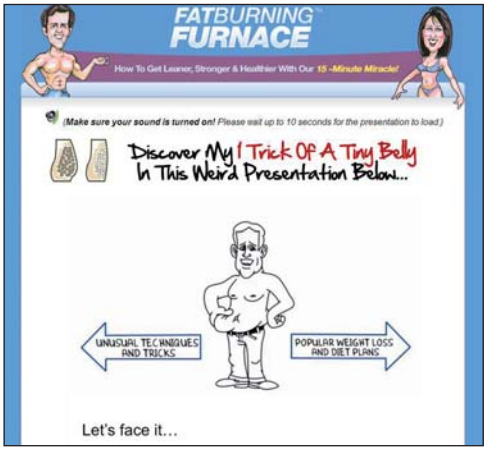

Figure 6: Fake health-related ads.

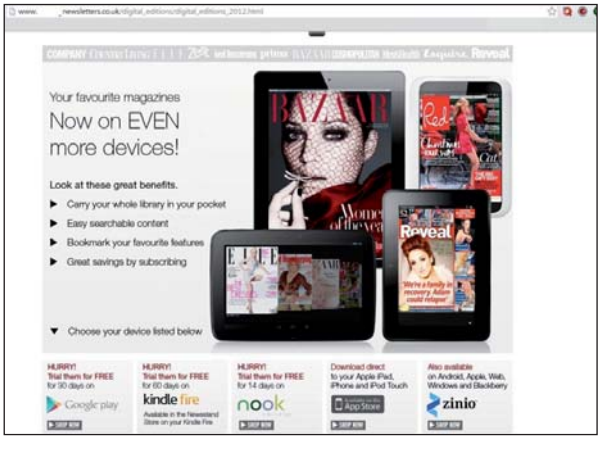

Figure 7: Newsletter services.

Figure 8: File sharing.

Figure 9: Games-related sites.

6. File sharing: 5.54%

This category includes web pages that allow users to share or store files online. Some dubious websites promoted through advertising are used for fraud, identity theft or malware infections, after luring users with fictitious software downloads (Figure 8).

7. Pornography: 5%

This category typically covers websites containing erotic and pornographic content (text, pictures or video). Accurate blacklisting may also detect erotic content on mixed websites classified in multiple categories.

8. Games: 4.78%

This category covers websites with games presentations, reviews, and online games including *Flash*-based applications (Figure 9). It also includes websites that offer the possibility of buying or downloading non-browser-based games. Non-legitimate games websites promoted through malvertising lead users to fake downloads and fraudulent surveys.

9. Illegal: 2.25%

This category covers websites related to software piracy, including peer-to-peer and tracker websites known as distribution channels for copyrighted content, pirated commercial software websites and discussion boards, as well as websites providing cracks, key generators and serial numbers for illegal software use (Figure 10). Illegal websites promoted through advertisements may lead users to malicious downloads and fraudulent URLs.

10. Job search: 2.02%

This category covers websites presenting job boards, job-related classified ads and career opportunities, as well as aggregators of such services. In the case of malvertising and other e-threats, the most common traps are job scams that ask users for money and personal details so they can follow their 'American dream' (Figure 11).

Figure 10: Illegal downloads.

Figure 11: Job scams encourage users to follow their 'American dream'.

11. Online shops: 1.76%

This category includes online stores and platforms that sell different goods or services. The typical threat infiltrating legitimate ad networks is once again fraud – scammers put up fake shops, register them on reputable TLDs such as '.com', and even design them better than some authentic ones. Poorly crafted online shops may also be breached by phishers, who can then steal users' money and sensitive data.

Figure 12: Fake shops.

12. Online dating: 0.74%

This category typically covers paid or free websites where users register to find a dating partner or a new relationship (Figure 13). An extension of the social networks category, online dating is typically misused to steal users' personal details and to help social engineers craft human databases.

13. Financial (banks): 0.73%

This category covers the websites of all banks and financial institutions, including loan companies, credit card agencies, and companies in charge of brokerage of securities or other financial contracts (Figure 14). One recent scam promoted through malvertising was a fake loan website that received almost 200,000 *Facebook* likes.

14. Travel: 0.25%

This category covers websites that offer travel facilities and equipment as well as destination reviews and ratings. Anti-fraud technologies blacklist fake websites in this

Figure 13: Online dating.

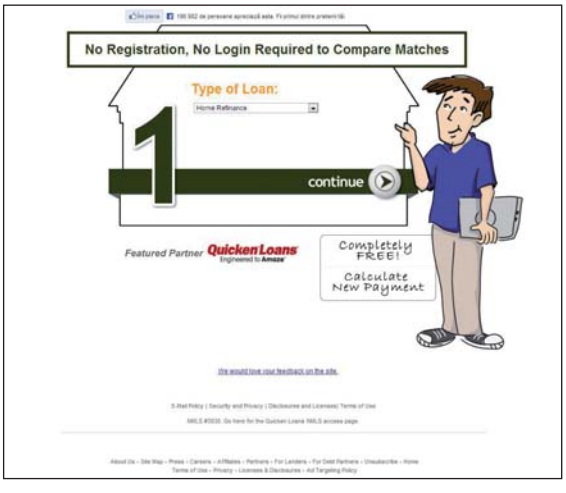

Figure 14: Financial services.

category if they discover they have been registered to trick users with fraudulent offers. Recently, a Dubai promotion website was being advertised through malicious techniques, as shown in Figure 15.

15. Portals 0.19%

This category covers websites that aggregate information from various sources and domains. Portals may also offer features such as search engines, email, news and entertainment information. Figure 16 shows a recent malvertised portal.

16. Instant messaging: 0.08%

This category covers websites where users can chat in real time or download IM software. Such websites may be

Figure 15: Dubai promotion website being advertised through malicious techniques.

Figure 16: Recent malvertised portal.

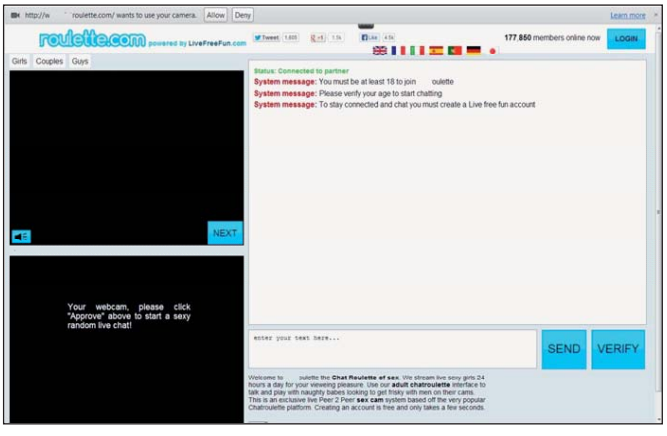

Figure 17: Instant messaging.

included in several other categories, such as gambling (see Figure 17).

17. Webmail (0.06%), entertainment (0.04%), social networks (0.04%)

These three less popular malvertising categories cover websites that provide email functionality as web applications, websites that provide information related to artistic activities, and social media websites.

COUNTRIES OF ORIGIN

Our research revealed the top countries of origin for malvertising websites. Most malvertising and other e-threats originate from the US, the Netherlands and Canada. This doesn't necessarily mean that the cybercriminals are residents of those countries, as they often register websites remotely in order to hide from law enforcement.

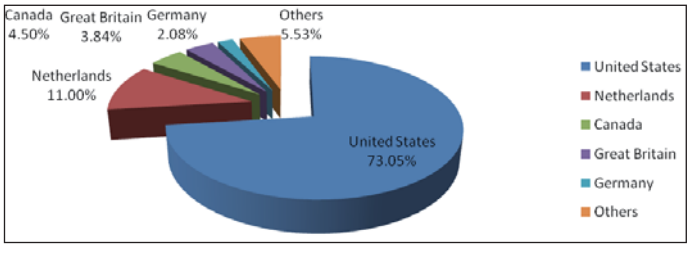

Figure 18: Countries of origin for malvertising and other e-threats.

GUIDELINES

To lessen the chances of being tricked by malicious and fraudulent advertisements, users and ad networks can take several precautions:

- Before activating commercials on their websites, companies and ad networks should carefully check their origin and legitimacy.
- To verify if a website is authentic, users and companies can look for *Whois* information to see if the web page is hosted in the country in which the company is based (a domain being registered in a different country may be a sign of a non-legitimate site). Most fraudulent websites are registered for just a year, which can also be a sign of a scam. Also, if a website has been registered to a private email address such as contact@privacyprotect.org or a webmail address such as john@yahoo.com, it's almost certainly not legitimate.
- To help mitigate the effects of malvertising, keep your anti-virus protection updated, together with your operating system and other software. Malicious files have less chance of being downloaded successfully if security solutions are installed and up to date.

CONCLUSIONS

Our research showed that almost 7% of landing pages multiplying on advertising platforms pose serious dangers to computer users. The most dominant malicious category is fraud, which takes the form of fake software offers, business and financial scams. In addition, users are targeted by malware injected into legitimate ads. By using their baits on multiple advertising platforms, scammers increase their chances of making money with minimal effort.

We also found that the highest concentration of online security threats promoted through advertising are found not among pornography and online dating sites, but on business and computer websites.

In conclusion, users and other stakeholders should be careful when dealing with online advertisements. If the inner structure of the system continues to be this open, with so many parties involved and without firm security scanning, cybercriminals will take advantage of companies, advertising platforms and end-users increasingly often.

REFERENCES

[1] Online Trust Alliance. Anti-Malvertising Resources. https://otalliance.org/resources/malvertising.html.

[2] Voluntary Anti-Malvertising Guidelines & Best Practices Helping to Combat Malvertising and Preserve Trust in Interactive Advertising. https://otalliance.org/docs/OTA_guidlines_final10_18.pdf.

[3] Damian, A. Understanding the domains involved in malicious activity on Facebook. Virus Bulletin, June 2012, p.19. http://www.virusbtn.com/virusbulletin/archive/2012/06/vb201206-Facebook.

[4] Cisco 2013 Annual Security Report (ASR). https://grs.cisco.com/grsx/cust/grsCustomerSurvey.html?SurveyCode=6653&KeyCode=000204094.

[5] Mlot, S. Online Advertising More Likely to Spread Malware Than Porn. http://www.pcmag.com/article2/0,2817,2415009,00.asp.

[6] Dima, B.; Damian, A. Phishing and fraud: the make-believe industry. Virus Bulletin, April 2013, p.33. http://www.virusbtn.com/virusbulletin/archive/2013/04/vb201304-phishing-fraud.

SPOTLIGHT

GREETZ FROM ACADEME: MONKEY VS. PYTHON

John Aycock
University of Calgary, Canada

Some programming languages have an embarrassment of riches when it comes to code obfuscation. For JavaScript, of course, every few months sees a fresh analysis of malicious code, such as Peter Ferrie's recent breakdown of JS/Proslikefan [1]. For C, code obfuscation is sport, with the International Obfuscated C Code Contest [2]. And Perl is… Perl is another programming language.

Python, however, has largely eluded the obfuscation craze. There are a few examples, including a beautiful Mandelbrot set generator whose code is shaped like a Mandelbrot set [3]; another post by the same author [4] contains links to some other scattered Python obfuscation examples, and there was a 2011 PyCon talk on the subject [5]. In the unlikely event that the bad guys ever decided to forsake JavaScript for Python, these few examples could turn out to be Useful Information.

All this means that when I see anything relating to Python obfuscation, it quickly gets my attention. That was the case with a paper from the 2013 USENIX Workshop on Offensive Technologies, called 'Looking inside the (Drop) box' [6], in which the authors detail their techniques for reverse engineering a 'hardened' Python application from *Dropbox*. It's a paper that wouldn't be out of place in the pages of *VB* and, much to my surprise, it turns out that I (very indirectly) helped with the work.

In the *Dropbox* case, the Python obfuscation was not at source level, but in the 'frozen' version that was shipped out. A frozen Python application is one where all the pieces of compiled Python bytecode are bundled together to allow a single file to be distributed. It's essentially a form of (non-malicious) packing, and a number of legitimate tools/scripts exist for this purpose – one even in the Python source distribution itself.

Dropbox's frozen executable was modified to make reversing it more challenging, though [6]. The opcode values were altered, the code was encrypted, and the normal means to query bytecode were removed, amongst other things. The researchers ended up injecting a DLL into the *Dropbox* process to gain control, allowing them eventually to inject their own Python code into the hardened interpreter. A few steps later (all of which are detailed in the paper), they had acquired the Python bytecode.

Once the bytecode had been extracted, the authors used a tool called uncompyle2 to reconstitute the Python source code. Upon further examination [7], I discovered that the tool is based on a Python compilation framework I created, and a Python decompiler that I cobbled together in around 1999. It's a small world, and it's reassuring, as an academic, to know that occasionally something useful comes of my work.

Back to the reverse engineering: after the Python source code had been extracted, the researchers worked around *Dropbox*'s authentication and gathered up SSL data, using a technique they called 'monkey patching'.

I must confess that I had never heard that term before, and it brought to mind either a roomful of monkeys with typewriters working on Shakespeare v2.0, or animals prone to flinging their own faeces. In neither case did it cast the reverse engineering in a terribly flattering light. Naturally, I turned to the arbiter of all that is true, *Wikipedia*, which helpfully informed me [8] – and I am not making this up – that the technique 'has also been termed **duck punching** and **shaking the bag**'. The emphasis is theirs, believe me. So while monkey patching sounds bad, the alternatives are even more ghastly. But I digress.

Apparently, monkey patching is simply poking into a dynamic language at run time and modifying things. This allowed the paper's authors to hook all the SSL objects in the Python code and dump out their data unencrypted. And thus *Dropbox* fell.

No monkeys, pythons, or ducks were harmed in the creation of this article.

REFERENCES

[1] Ferrie, P. Fans like pro, too. Virus Bulletin, September 2013. http://www.virusbtn.com/vba/2013/09/vb201309-Proslikefan.

[2] The International Obfuscated C Code Contest. http://www.ioccc.org/.

[3] Preshing, J. High-resolution Mandelbrot in obfuscated Python. http://preshing.com/20110926/high-resolution-mandelbrot-in-obfuscated-python/.

[4] Preshing, J. Penrose tiling in obfuscated Python. http://preshing.com/20110822/penrose-tiling-in-obfuscated-python/.

[5] Healey, J. How to write obfuscated Python. PyCon 2011. http://blip.tv/pycon-us-videos-2009-2010-2011/pycon-2011-how-to-write-obfuscated-python-4899191.

[6] Kholia, D.; Wegrzyn, P. Looking inside the (Drop) box. 7th USENIX Workshop on Offensive Technologies, 2013.

[7] uncompyle2/README. Uncompyle2 0.13, 22 February 2012. Source available on GitHub.

[8] Monkey patch. http://en.wikipedia.org/w/index.php?title=Monkey_patch&oldid=575983320.

END NOTES & NEWS

The 22nd Annual EICAR Conference takes place 17–19 November 2013 in Hannover, Germany (postponed from earlier in the year). For full details see http://www.eicar.org/.

Oil and Gas Cyber Security will be held 25–26 November 2013 in London, UK. For details see http://www.smi-online.co.uk/2013cyber-security5.asp.

Black Hat Regional Summit takes place 26–27 November 2013 in Sao Paolo, Brazil. See http://www.blackhat.com/sp-13/.

AVAR 2013 will take place 4–6 December 2013 in Chennai, India. For details see http://www.aavar.org/avar2013/.

Botconf 2013, the 'first botnet fighting conference', takes place 5–6 December in Nantes, France. For details see https://www.botconf.eu/.

Black Hat West Coast Trainings take place 9–12 December 2013 in Seattle, WA, USA. For details and registration see https://www.blackhat.com/wc-13/.

FloCon 2014 will be held 13–16 January 2014 in Charleston, SC, USA. For details see http://www.cert.org/flocon/.

The 6th International Forum on Cybersecurity takes place 21–22 January 2014 in Lille, France. For more information see http://www.forum-fic.com/2014/en/.

RSA Conference 2014 will take place 24–28 February 2014 in San Francisco, CA, USA. For more information see http://www.rsaconference.com/events/us14/.

Cyber Intelligence Asia 2014 takes place 11–14 March in Singapore. For full details see http://www.intelligence-sec.com/events/cyber-intelligence-asia-2014.

Black Hat Asia takes place 25–28 March 2014 in Singapore. For details see http://www.blackhat.com/.

SOURCE Boston will be held 9–10 April 2014 in Boston, MA, USA. For more details see http://www.sourceconference.com/boston/.

The Infosecurity Europe 2014 exhibition and conference will be held 29 April to 1 May 2014 in London, UK. For details see http://www.infosec.co.uk/.

The 15th annual National Information Security Conference (NISC) will take place 14–16 May 2014 in Glasgow, Scotland. For information see http://www.sapphire.net/nisc-2014/.

SOURCE Dublin will be held 22–23 May 2014 in Dublin, Ireland. For more details see http://www.sourceconference.com/dublin/.

The 26th Annual FIRST Conference on Computer Security Incident Handling will be held 22–27 June 2014 in Boston, MA, USA. For details see http://www.first.org/conference/2014.

Black Hat USA takes place 2–7 August 2014 in Las Vegas, NV, USA. For details see http://www.blackhat.com/.

VB2014 will take place 24–26 September 2014 in Seattle, WA, USA. More information will be available in due course at http://www.virusbtn.com/conference/vb2014/. For details of sponsorship opportunities and any other queries please contact 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/