ARTICLE:

KNOWN KNOWNS, KNOWN UNKNOWNS AND UNKNOWN UNKNOWNS: ANTI-VIRUS ISSUES, MALICIOUS SOFTWARE AND INTERNET ATTACKS FOR NON-TECHNICAL AUDIENCES

By **Daniel Bilar**

Introduction

The risks associated with the internet have changed significantly. A recent study claims that a typical Microsoft Windows machine is subjected to autonomous infiltration attempts – not just mere pings and probes – from worms and botnets looking for clients once every six minutes.[1] Stealth – not exhibitionism or hubris – characterizes this breed of attacks and concomitantly deployed malicious software. Unbeknownst even to experienced human operators, surreptitious attacks are able to insert malicious code deep within the bowels of individual computers and the wider supporting internet communication and control infrastructure such as wireless access points, home routers, and domain name servers.[2] In addition to stealth, social engineering via e-mail, Instant Messaging, and social networks plays an important part, as well: unsuspecting users are coaxed to initiate actions that infect their computers and usurp their digital identities.

These attacks are powerful because of the havoc that is causes to the owner or user of the computer or computer network. The effects range from mere nuisance, to the appropriation of sufficient information to impersonate an individual, that can, in turn, lead to financial ruin, up to and including criminal charges against the innocent.[3] They are also powerful because, in many instances, neither individual computer owners, nor the sophisticated network controlled by a government can prevent all of the malicious code from penetrating their computers or networks.

In the past, the actions of hobbyists and isolated mischief makers merely caused disruptions. Now organized and highly technically competent criminals with financial incentives as the primary motivator have taken over. In addition, semi-independent state-sponsored groups occasionally launch attacks on another state. The ramifications of this shift are worrisome: that a person may subscribe to an anti-virus software product from one of the many vendors on the market does not mean that their computer is protected from or necessarily free of malicious software.

Modern malicious software has been shown in tests carried out in independent laboratories to be highly resistant to being identified by anti-virus (AV) products. In addition, these empirical results are consistent with theoretical findings, in that detecting complex malicious software is beyond the effective modelling capabilities of current AV products,[4] and as such is becoming

[1] *Gabor Szappanos, 'A Day in the Life of An Average User', Virus Bulletin, January 2009, 10-13, available at http://www.virusbtn.com/.*

[2] *Most users do not bother to change the default passwords on home devices such as routers. Browser vulnerabilities can then be exploited by malicious software to alter the DNS settings of the router, thereby directing any name lookup query to a DNS of the attacker's choice. This may be used to spoof a bank web site, for instance. See Sid Stamm, Zulfikar Ramzan and Markus Jakobsson, 'Drive-By Pharming', Lecture Notes in Computer Science 4861, (Springer, 2007), 495-506 and Hristo Bojinov, Elie Bursztein, Eric Lovett and Dan Boneh, 'Embedded Management Interfaces: Emerging Massive Insecurity', Blackhat Technical Briefing, Blackhat USA 2009 (Las Vegas, USA, August 2009), available at http://www.blackhat.com/presentations/bh-usa-09/BOJINOV/BHUSA09-Bojinov-EmbeddedMgmt-PAPER.pdf.*

[3] *For examples of people charged with offences, see Patrick v Union State Bank, 681 So.2d 1364 (Ala. 1995); Vic Lee, 'ID Theft Puts Innocent Man In San Quentin', 21 February 2007, ABC7News, available at http://abclocal.go.com/kgo/story?section=news/local&id=5052986 and Mary Pat Gallagher, 'Identity-Theft Victims Owed Duty of Care in Bank Fraud Investigations, N.J. Court Says', Law.com, 11 September 2008, available at http://www.law.com/jsp/article.jsp?id=1202424426977.*

[4] *Yingbo Song, Michael E. Locasto, Angelos Stavrou, Angelos D. Keromytis and Salvatore J. Stolfo, 'On the infeasibility of modelling polymorphic shellcode,' Proceedings of the 14th ACM conference on Computer and Communications Security, 2007, 541–551.*

increasingly difficult to detect in practice, and worryingly, also in principle.[5] To put it simply, anti-virus software does not prevent all forms of malicious software from penetrating computers and networks – some malicious software will not be identified by anti-virus software, which is why this is an important topic for lawyers and judges to understand.

The aim of this article is to introduce the technical issues surrounding modern internet attacks, anti-viral software and malicious software to the individual that has no technical knowledge, and who needs a working understanding of the pertinent issues. As such, its primary goal is to raise awareness, not comprehensiveness. The interested reader is referred to a recent book by Markus Jakobsson and Zulfikar Ramzan, *Crimeware. Understanding New Attacks And Defenses*, (Symantec Press, 2008) for further study.

Software vulnerabilities

Coding errors[6] in software can lead to vulnerabilities. Software vulnerabilities are program weaknesses which malicious software can exploit. The relationship between coding errors, vulnerabilities and exploitation is illustrated by the following analogy: the US Tariff Act of 1872 was to include a list of duty-free items: Fruit plants, tropical and semi-tropical. A government clerk duly transcribed the Act, but erroneously moved the comma: Fruit, plants tropical and semi-tropical. Shrewd businessmen argued that the law, as promulgated, exempted all tropical and semitropical plants from duty fees, resulting in $500,000 loss to the US Treasury.[7] For the purposes of this discussion, the erroneous placement of the comma is the equivalent of a software coding error. The vulnerability resulting from this error manifests itself as an opportunity for alternative interpretation, and the exploit is represented by cleverly taking advantage of duty-free imports of tropical and semi-tropical plants.

Since errors in software coding errors permit malicious exploitation, it seems obvious that efforts should concentrate on writing error-free code. Unfortunately, industrial software has exhibited the same code error density for the past twenty years; on average six faults (errors) for every thousand lines of source code.[8] However, the general increases in the amount of code (Windows Vista has an estimated 80 million lines, whereas Windows 2000 had 35 million lines), as well as the complexities of modern software (interactions between components and protocols, as well as very large applications like Adobe Acrobat Reader with 2 million lines of code) have exacerbated the situation.

The survival time of an unpatched Windows system may serve as corroborating evidence.[9] In 2003, an unpatched Windows PC would last approximately 40 minutes on average, before it would succumb to probes from (presumably) malicious software. In 2004, survival time was reduced to 16 minutes and by 2008, the time window had shrunk to mere 4 minutes.[10]

Just as a motor car needs regular tune-ups to keep running smoothly, maintenance of installed software is performed through regular updates. Since software vulnerabilities are the root cause of many malicious software infections, updating (or equivalently patching) minimizes the number and severity of software vulnerabilities that malicious software may exploit. The poor quality of software code explains in part why anti-virus software is required in the first place (another factor is ubiquitous connectivity). Anti-virus software, however, has problems of its own.

The problem with anti-virus software

Most commercial AV products rely predominantly on some form of signature matching to identify malicious code. In the context of AV, a signature is the rough software equivalent of a fingerprint – it is a pattern that identifies malicious software. It is possible to derive a pattern from software code, that is, a static snippet of code (or a uniquely reduced version of it, such as a hash). The fragment is taken as the pattern that identifies the code. A static signature is, in its simplest incarnation, a fixed sequence of characters somewhere

[5] *Grégoire Jacob and Eric Filiol and Hervé Debar, 'Malware as interaction machines: a new framework for behavior modelling,' Journal in Computer Virology, Volume 4, Number 3, August 2008, 235-250.*

[6] *For an overview of such errors, see Katrina Tsipenyuk, Brian Chess and Gary McGraw, 'Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors', IEEE Security and Privacy, Volume 3, Issue 6, (November 2005), 81-84.*

[7] *See 'Forty-Third Congress; First Session Feb. 20', New York Times, February 21, 1874, at http://query.nytimes.com/mem/archive-free/pdf?res=9902EFD8173BEF34BC4951DFB466838F669FDE.*

[8] *Compare John Musa, Software Reliability Measurement Prediction Application (McGraw-Hill, 1987) with Parastoo Mohagheghi and Rediar Conradi, 'An empirical investigation of software reuse benefits in a large telecom product', ACM Transactions on Software Engineering Methodology, Volume 17, Issue 3 (June 2008), 1-31.*

[9] *A patched system denotes a computer on which the latest software updates (normally for the Operating System, but also for Office suites and media software) have been installed.*

[10] *See John Leyden, 'Unpatched Windows PCs own3d in less than four minutes', The Register, 15 July 2008 at http://www.theregister.co.uk/2008/07/15/unpatched_pc_survival_drops/ and Survival Time at http://isc.sans.org/survivaltime.html.*

in a file or in memory and may look something like this:

```
C3 7C FD 1D 31 C0 6F OF 96 18 A4
```

The rationale underlying these character patterns is that they are more likely to be encountered when analyzing malicious software rather than innocent programs. Hundreds of thousands of these signatures are stored in local AV databases (AV signature updates are received, hopefully, at least once a week). An AV scanning engine then tries to match pre-defined file areas against this signature database. These areas are typically located at the beginning and the end of the file, and after what is called the executable entry point of a program.

Strict matching of the byte sequence pattern was most popular in the early 1990s. This method has since been augmented, because those responsible for writing malicious code took action to avoid being noticed by the AV products. They approached their evasion in a straightforward way. Because of time constraints (users tend not to wait more than a couple of seconds), it is not usual to scan the whole file. Malicious authors took advantage of this fact and moved the malicious code to locations in the file that would probably not be scanned. Furthermore, they tweaked their malicious code to make the byte pattern mismatch. One way of doing this is by equivalent instruction substitution. An example will illustrate this point. In the signature above, the substring pattern `31 C0` represents Intel machine code `xor ax, ax`. Its purpose is to set register `ax` to `0`. A substitution that preserves this functionality would replace the substring with `29 C0` (which is machine code for `sub ax, ax`) or `B8 C0 00` (which is machine code for `mov ax,0`).

Generic matching was introduced to add some 'fuzziness' to the signature in order to catch malicious software that is slightly altered so as to evade the stricter matching. Using the example above, the second, third, fourth and ninth bytes are replaced with a wildcard (a 'blank', do-not-care byte) denoted by '??':

```
C3 ?? ?? ?? 31 C0 6F OF ?? 18 A4
```

When searching for this pattern, the '??' directs the AV scanner to ignore whatever byte value is present in the second, third, fourth and ninth bytes of character strings it encounters while scanning the file. For example the string below:

```
C3 99 A0 BB 31 C0 6F OF 77 18 A4
```

would match, as well as:

```
C3 A1 22 00 31 C0 6F OF FF 18 A4
```

Hence, wildcards try to lower AV false negative detection rates by 'softening' the signatures to counteract some of the evasive coding tactics that malicious software is programmed to use to avoid detection. The problem with casting a wider net to catch 'bad' programs is that 'innocent' (that is non-malicious) programs may be identified incorrectly; in other words, there is an increase in the false positive rate.

For an accessible overview of more AV signature detection enhancements, the reader is encouraged to peruse chapter 11 of Peter Szor, *The Art of Computer Virus Research and Defense*, (Addison Wesley, 2005).

Static signatures, as we have discussed them so far, are derived from program code, reflecting the byte value make-up of a program. Malicious software detection at the beginning of the twenty-first century started to incorporate *behavioural heuristics* approaches; that is, a notion of how a given software program interacts with its embedded environment. For instance, a program may interact with a file system (by opening, creating or deleting a file), or the network (opening a connection to a server or setting up a receiving server). These and other interactions of the program can be monitored in what is called a 'sandbox'. A sandbox is a controlled, instrumented container in which the program is run and that records how it interacts with its environment. A sample sandbox output is set out below:

```
[ General information ]
* Display message box (sample) : sample, tikkun olam!
* File length: 18523 bytes.
* MD5 hash: 1188f67d48c9f11afb8572977ef74c5e.
```

Here some general information about the file (its length and its hash) is made visible, together with what is displayed on screen (a message box with the caption 'sample' and message 'tikkun olam!'). The next phase is for the malicious software to carry out instructions to delete a file and place a substitute file in place of the file that has been deleted:

```
[ Changes to filesystem ]
* Deletes file C:WINDOWS\SYSTEM32\kern32.exe.
* Creates file C:WINDOWS\SYSTEM32\kern32.exe.
```

Here we see that the first action of the program is to delete a file and recreate one with the same name, kern32.exe. This is suspicious. Then it is necessary to enter the internal Windows database (the Windows registry). This is illustrated below. This entry makes the file kern32.exe run when system startup begins as the computer is switched on:

```
[ Changes to registry ]
* Creates key
"HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce".
* Sets value "kernel32"="C:WINDOWS\SYSTEM32\kern32.exe -
sys" in key " HKLM\Software\ Microsoft\Windows
\CurrentVersion\RunOnce".
```

This is very suspicious behaviour, in that the system is instructed to intercept the strokes used on the keyboard and pass it on to a custom function:

```
[ Changes to system settings ]
* Creates WindowsHook monitoring keyboard activity.
```

There follows the network activity: the program connects to a server at address 110.156.7.211 on port 6667, a typical port for Internet Relay Chat (IRC) chat server, logs in and joins a chat channel:

```
[ Network services ]
* Connects to "110.156.7.211" on port 6667 (TCP).
* Connects to IRC server.
* IRC: Uses nickname CurrentUser[HBN][05].
* IRC: Uses username BoLOGNA.
* IRC: Joins channel #BaSe_re0T.
```

In the example above, interactions occur with the file system, the Windows registry (the internal Windows database) and the establishment of a TCP network connection to an IRC chat server. Connecting to a chat server is anomalous enough behaviour that it should raise a concern that something is not correct. Taken together, this set of activities is consistent with the suspicious program being a bot, connecting to a botnet through the IRC server.

Thus, behavioural heuristics seek to establish an 'activity' profile. It is also possible to derive a 'behavioural signature' from such an activity profile (as opposed to the byte-value approach discussed earlier).[11]

Just as there are different ways of rewriting instructions (as seen with the `xor ax,ax` example above), there are ways of effecting the same or similar behaviour: a Windows program may open a file by means of user mode API `NtOpenFile()/OpenFile()`, kernel-mode API `ZwOpenFile()` or may even bypass the API completely and directly access the disk driver with `IoCallDriver()` with manually constructed IO packets. How well these signatures approaches work in practice will be discussed below.

Practical AV concerns: false negatives

A number of independent laboratories regularly test updated AV scanners against millions of malicious software specimens. These scanners predominantly use byte-value signature approaches, though almost all of them today incorporate some form of (much slower) behavioural detection. Some empirical data for sixteen well-known, reputable AV products are shown in Table 1.

Report Date	AV Signature Update	MW Corpus Date	False Negative (%)	Scan Speed (MB/sec)
2009/05	Feb. 9th	Feb. 9th -16th	[31-86]	N/A
2009/02	Feb. 9th	Feb. 1st	[0.2-15.1]	[24.0-3.7]
2008/11	Aug. 4th	Aug. 4th -11th	[29-81]	N/A
2008/08	Aug. 4th	Aug. 1st	[0.4-13.5]	[22.2-2.9]
2008/05	Feb. 4th	Feb. 5th -12th	[26-94]	[25.5-1.6]
2008/02	Feb. 4th	Feb. 2nd	[0.2-12.3]	N/A

Table 1: Miss rates of up-to-date scanners.
Table generated by the author from AV-comparatives.org data

The reader is requested to note how quickly AV signature databases go out-of-date. After failing to update signatures for one week, the *best* AV tested missed between 26 and 31 per cent of the new malicious software, the worst missed upwards of 80 per cent. The empirical test results from http://www.av-comparatives.org/comparatives reviews indicate that the claims made by vendors of AV products must be soberly assessed.

There is preliminary hope pinned on 'cloud computing' environments, where vendors promise reactive signature generation times on the order of

[11] *For a review of behavioural based scheme and a recent prototype of a behaviour based signature approach (using a system-call-data flow dependency behaviour graph), see Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and Xiaofeng Wang, 'Effective and Efficient Malware Detection at the End Host', in USENIX Security '09, Montreal, Canada, (August 2009), available at http://www.iseclab.org/publications.html.*

The ability to disguise malicious software becomes more subtle and unpredictable in the light of the different methods by which devices now communicate with each other.

minutes, not days, through active internet connections. This remains to be seen, as the race between AV companies and malicious software writers continues.

The problem with modern malicious software

Modern malicious software is interactive, polymorphic and metamorphic. All these terms have to do with the methods used to bypass the approach used by AV products to detect malicious software (and other signature-based defences such as intrusion detection systems). Polymorphism and metamorphism are both techniques to mutate the computer code of the malicious software while keeping its malicious functionality unchanged. The purpose of this is to evade the signatures of AV.

Though the terms are sometimes used interchangeably, there are technically different: polymorphic malware typically uses encryption on parts of its code containing its malicious functionality. This code must be decrypted by a decryptor routine before it can be executed. Typically, both the encryption and decryption loops can be identified (in unencrypted form) in the malicious software, although it is possible to out-source this function to a remote server – called server-side polymorphism. Hence, the main characteristics of truly polymorphic malicious software are the use of encryption and a fixed decryptor routine.

For detection purposes, encrypted code has a distinct general signature (it has high entropy because of the diffusion property of good encryption); as such, AV can discern the existence of encrypted code, if not its purpose or functionality. Benign code may also be encrypted (and commonly is for intellectual property reasons), thus limiting the usefulness of high entropy detection approaches. The fixed decryptor routine of truly polymorphic code can easily be picked up by byte-pattern signature-based AV. It is for this reason that writers of malicious software have devised schemes to generate mutated, but functionally equivalent decryptors in subsequent generations, leading to what is called 'oligomorphic' code. Oligomorphic decryptor mutation approaches in turn lead to the development of 'metamorphic' code.

Metamorphic code strives to change its appearance from generation to generation, whilst ensuring that it continues to function as it was designed to. Metamorphic malicious software typically does not use encryption. Instead, it is written in such a way that it attempts to re-arrange the relative position of its code, substitute certain instructions, register re-assignments, changes sequence permutation and uses other substitution or permutation techniques. Part of the malicious code incorporates a metamorphic engine that performs these alterations, or the malicious software contacts a server for the task. If the latter, it makes detection harder. Similarly to the truly polymorphic case, a transformation engine residing in the code offers more opportunities for detection purposes.

The ability to disguise malicious software becomes more subtle and unpredictable in the light of the different methods by which devices now communicate with each other. In the widest sense, almost any form of external input might cause malicious software to become active or, more distressingly, provide a missing piece of code to turn apparently innocuous fragments of code into malicious software. Time is used as a mechanism to cause malicious software to become active through internal system clocks (the 1992 Michelangelo virus was activated in this way on the anniversary of his birthday, 6 May), and human activity in using the computer, opening a file or browsing a website can also be used to activate malicious software.

As previously noted, the problem of identifying malicious software is also exacerbated because of ubiquitous connectivity. The vast majority of computers are constantly interacting over the network (end users have little choice in the matter, because software licenses tend to be remotely attested), and at any

moment, passive (as in a simple packet) and active (as in code) prompts can be added to the recipient's system with no prior indication of what this single piece of code will induce. A recent example was provided by the fourth generation of the Conficker worm, when millions of people waited to see what the code would do on the 1 April 2009.[12]

Theoretical AV concerns: detection complexity

As the empirical results suggest, meta- and polymorphic coding techniques pose an aggravated detection challenge for AV. In addition, malicious software has become increasingly modular (utilizing ubiquitous connectivity), and exhibits what is called 'staged downloads'. Staged downloads involve an initial compromise in which a small piece of code is installed. This is effected, for instance, by an network worm exploiting an Operating System vulnerability (such as the 2008 Gimmiv.A worm that targeted the Windows MS08-67 vulnerability[13]) and depositing an initial payload. It could also be effected by the user opening e-mail attachments with malicious code attached, and increasingly, through vulnerabilities in web browsers on computers. The initial infection is subsequently followed up with the installation of more malicious code to fulfil one or more of the objectives that the code is designed to carry out (among them spam relay, stealing of personal information, industrial espionage). Almost 80 per cent of potential malicious code infection exhibit these staged downloads.[14]

Malicious code communicates with its environment for the purposes of propagation and to receive instructions and download new binary code. As a result, the AV detection problem becomes much more difficult. It becomes much harder (impossible in the general case) for AV to decide whether fragments of code are malicious, since not all the pieces may have been assembled. The changing dynamics of malicious code and how it is created and disseminated (complete or in small pieces, and then assembled), means that reliable detection cannot realistically be achieved within time constraints of seconds, if it can be done at all.

Anti-virus: epilogue

Because of the metamorphic and polymorphic dissimulation techniques and the modular staged downloads, current AV is not able to ascertain (within acceptable false negative rates and time limits, and sometimes not in principle) whether code is malicious or not. Worse still, the methods by which an individual can inadvertently download malicious software not only include programs that might be explicitly installed, but code that is installed and executed surreptitiously from a visit to perfectly respectable websites, unbeknownst to the user. The TDSS rootkit serves as an informative case study that demonstrates how malicious software is capable of being installed in seemingly innocuous parts (in the form of a legitimate but maliciously patched DLL) which enables the subsequent downloading and execution of any other arbitrary (malicious) component. Of further interest are the multiple methods of infection used to infect a system (including website vulnerabilities, peer-2-peer networks, video viewing and other software).[15]

The user is faced with stark choices, none of which mitigate the effects of these threats completely. She may disable the functionality that makes web browsing a rich experience to minimize the risk of attack. This means, in effect, reverting back to using the web with 1995 technology.[16] The user might decide to set up a virtual environment that enables the computer to be returned back to a known un-infected state, though this demands a level of discipline that few users are capable of. The Google Chrome web browser is a step in this direction. It incorporates a light-weight virtualized environment called GreenBorder that sets up a protected browser that seeks to shield the computer system from actions originating from browsing the internet.[17] The last choice is the worst and alas, the most common: taking a deep breath, clicking away and trusting anti-virus software to an extent that is not warranted.

Inviting attacks from the internet

There are indications that some safe computing procedures have begun to be understood by end users. For instance, many users now know better than to open e-mail attachments, and they are more mindful of keeping their system patches and AV signatures up-to-date. These measures offer some limited protection.

[12] *For a recent, sophisticated example of binary code updates that is encrypted and electronically signed, see Phillip Porras, Hassen Saidi, and Vinod Yegneswaran, An Analysis of Conficker's Logic and Rendezvous Points, (SRI International Technical Report), 2009 at http://mtc.sri.com/Conficker/ and http://mtc.sri.com/Conficker/addendumC/index.html.*

[13] *See http://www.microsoft.com/technet/security/Bulletin/MS08-067.mspx for the vulnerability and http://www.f-secure.com/v-descs/trojan-spy_w32_gimmiv_a.shtml for a description of the worm.*

[14] *For which, see Symantec's annual Global Internet Threat Report at http://www.symantec.com/business/theme.jsp?themeid=threatreport.*

[15] *Alisa Shevchenko, 'Case Study: The TDSS Rootkit', Virus Bulletin, May 2009, 10-14.*

[16] *One example of a text-only web browser is lynx (http://lynx.isc.org/).*

[17] *GreenBorder was bought by Google in 2007.*

However, the act of browsing the web is more fraught with danger than commonly assumed. Web clients are now increasingly used for banking, health care, governmental services, and retail shopping from the comfort of one's home. Contemporary browsers, such as Internet Explorer, Opera and Firefox incorporate more functions than the mere display of text and images, including rich dynamic content comprising media playback and interactive page elements such as drop-down menus and image roll-overs. These features includes web browser extensions such as Javascript programming language, as well as additional features for the browser, such as application plugins (Acrobat Reader, QuickTime, Flash, Real, and Windows Media Player), and Microsoft-specific enhancements such as Browser Helper Objects and ActiveX (Microsoft Windows's interactive execution framework). Some of these extensions have security vulnerabilities that can maliciously exploited (ActiveX, Flash, and QuickTime make up the vast majority of plug-in vulnerabilities),[18] some are general programming languages or environments that can be tampered with for malicious purposes.

The fundamental issue is one of trust. When the user goes to a website from his browser, he types in a URL, and initiates the connection. Assume the user is visiting an on-line merchant, and assume an encrypted HTTPS connection is established (which is considered 'safe' browsing). The user logs on with his name and password, and a cookie is created. This cookie stores user preferences, such as session information and information about what the customer has purchased, and this cookie is typically placed on the user's computer. Once connected, a relationship of trust is established: the user and the website (the user initiated the connection, and now trusts the page and content display) and conversely, the site and the user (in executing actions from the user's browser). It is this trust, together with the various features incorporated into the browser that attackers try to subvert through what is called Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) attacks.

Cross-site scripting attacks mostly use legitimate web sites as a conduit, where web sites allow other (malicious) users to upload or post links on to the web site. Such links may contain malicious content (such as Javascript in obfuscated form) within them. They are then presented in an appealing manner ('Click here to view pictures!') to entice the victim to click on them. The malicious script in the link is executed in the victim's browser, and can copy cookie information, change user preferences, write information to files, or (in the form of a CSFR) obtain the data relating to log-ins to merchants and banks to perform actions that purport to be initiated by the customer. It is not only web servers can serve as a conduit: in 2005, a user named Samy Kamkar placed malicious Javascript on his MySpace profile. When a user viewed his profile, an XSS attack would add the user as a friend and place the malicious code in the viewer's profile. In twenty hours, over a million MySpace users were infected.[19]

Prevention of XSS attacks requires both server and client diligence. With respect to the server, software developed for web applications should check links posted by users for potentially malicious content, such as embedded Javascript and HTML code. Since code in such links would be executed in the browser of an innocent user, failure to validate (potentially malicious) input by users represents a software vulnerability that developers should address as a matter of course. Where users are concerned, they should exercise judicious care when clicking on a link. They may also take steps to be much less susceptible to XSS attacks. This can be accomplished by disabling JavaScript, Java, Flash, ActiveX and other dynamic content features in the browser. However, users will incur a severe usability penalty, since many websites depend on these features for to be viewed at their best.

Cross-site Scripting attacks are often used as a stepping stone with more insidious CSRF attacks in which a user's credentials are used for unauthorized transactions. For example, assume the victim is logged into a bank site. There are valid credentials, stored in form of a cookie on the victim's computer. The victim might casually surf a news site where an attacker was allowed to insert code of the sort illustrated below in a

[18] *In May 2009, the web-based Gumblar/JSRedir-R trojan (which accounted for over forty per cent of malicious content found on websites in the first week of May) used obfuscated JavaScript via web browsers to exploit vulnerabilities in Acrobat Reader and Flash Player. See Erik Larkin, 'New Wave of "Gumblar" Hacked Sites Installs Google-targeting Malware', PC World, May 14, 2009 at http://www.pcworld.com/article/164899/new_wave_of_gumblar_hacked_sites_installs_googletargeting_malware.html.*

[19] *See Justin Mann, 'MySpace speaks about Samy Kamkar's sentencing', TechSpot.com, January 31, 2007, where the following was noted: 'Samy Kamkar (aka 'Samy is my Hero') plead guilty yesterday in Los Angeles Superior Court to a violation of Penal Code section 502(c)(8) as a felony and was placed on three years of formal probation, ordered to perform 90 days of community service, pay restitution to MySpace, and had computer restrictions placed on the manner and means he could use a computer – he can only use a computer and access the internet for work related reasons' at http://www.techspot.com/news/24226-myspace-speaks-about-samy-kamkars-sentencing.html.*

It must be emphasized that from the point of view of the user, neither HTTPS (the encrypted channel with the little lock in the browser that denotes 'safety') nor logins protect against XSS or CSRF attacks.

posting or comment on the web site:

```
http://www.bankoflondon.com/transfer.p
hp?account=686868&amount=25000
```

If the attacker succeeds in inducing the victim to click on this link ('Click here to look at Michael Jackson's shroud' might work), a transaction request from the user's browser to the bank would be generated, attempting to transfer $25,000 to (presumably) the attacker's account number 686868.

Sometimes, it is not necessary to click on link on a web site. The news web site might contain HTML code (posted by the attacker) of the sort (purportedly to load an image) as illustrated below:

```
<img src=
"http://bankoflondon.com/transfer.php?
account=686868&amount=25000 width="1"
height="1" border="0">
```

With this code, the browser will try to load a miniscule image. This is a standard procedure to render images in web pages. But the image is not an image: it is actually a HTTP request to the fictional Bank of London, attempting to transfer $25,000 from the victim to the attacker's account number 686868. There is no image available, which means an error (crossed-out) image will be rendered by the browser to the user's screen. The reason for setting the size at 1 pixel by 1 pixel is to suppress this error image, and thus allay any suspicion of the victim.

The nature of transactions that are possible to effect depend on the site for which the credentials are valid. This can range from a posting to a message board with the user's identity; performing bank transactions, to changing the DNS settings of the home router (called drive-by-pharming) and buying stocks. In February 2008, 18 million users of an e-commerce site in Korea were affected by a CSFR attack.[20]

Similar to the XSS example, CSFR attacks can use other conduits (Adobe Acrobat, MS Word, RSS), provided these data formats allow for scripting. It must be emphasized that from the point of view of the user, neither HTTPS (the encrypted channel with the little lock in the browser that denotes 'safety') nor logins protect against XSS or CSRF attacks. In addition, unlike XSS attacks which necessitate user action by clicking on a link, CSFR attacks can be executed without the user's involvement, since they exploit explicit software vulnerabilities on the server. However, it is to be notes that CSFR attacks can be executed without the user's involvement, because they exploit explicit software vulnerabilities (predictable invocation structures) on the server. As such, it is suggested that the onus to prevent CSFR attacks falls squarely on the developers of such applications. Some login and cryptographic token approaches, if conscientiously designed to prevent CSFR attacks, can be of help.[21]

Epilogue

The wide variety of features that are included in everyday programs (such as web browsers and document viewers such as Adobe Acrobat Reader) are a serious concern: almost no user is aware that merely clicking on a URL, handling a PDF document[22] or simply

[20] *For which see 'WHID 2008-10: Chinese hacker steals user information on 18 MILLION online shoppers at Auction.co.kr' at http://www.webappsec.org/projects/whid/byid_id_2008-10.shtml.*

[21] *See the Secret Token scheme reviewed in Adam Barth, Collin Jackson and John C. Mitchell, 'Robust Defenses for Cross-Site Request forgery', in Proceedings of the 15th ACM Conference on Computer and Communications Security (Alexandria, Virginia, USA, October 27-31, 2008). CCS '08. ACM, New York, NY, 75-88, available from http://flyer.sis.smu.edu.sg/srg/ and http://www.adambarth.com/.*

[22] *See http://blog.didierstevens.com/2009/03/04/quickpost-jbig2decode-trigger-trio/ for an example where merely looking at PDF files in Windows Explorer (not opening them by double-clicking) launches the malware.*

surfing on to a webpage[23] may lead to a stealthy compromise and install powerful malicious software. As stated previously, a user has some protection against malicious software by keeping their system conscientiously patched, and maintaining up-to-date AV software. AV performs best if signatures are continuously updated, otherwise the practical detection rate plummets very quickly. Interactive malicious software, as well as user expectations,[24] significantly increases the difficulty of detection for AV software.

Hardware-based malicious code, which can be hidden in underlying integrated circuits (manufactured in China, and possibly compromised in the factory), will cause even more problems. Hardware subversion is not within the ability of AV software to deal with (in fact, it is an open research problem as to how to detect such malicious code in hardware at all). Hence, AV has its limitations, and care must be taken to ensure a digital evidence specialist, when examining a hard disk or live RAM memory, is aware of the various methods by which malicious software can be placed on a computer without the knowledge or authority of the owner or user.[25]

Daniel is an Assistant Professor in the Department of Computer Science, University of New Orleans, Louisiana, United States of America. He is a founding member of ISTS at Dartmouth College (NH, USA), conducting counter-terrorism critical infrastructure research for the US DoJ and US DHS. Active research topics include detection and containment of highly evolved malware and quantitative risk analysis and management of networks.

daniel@cs.uno.edu

[23] *Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu, 'The Ghost in the Browser: Analysis of Web-based Malware', Proceedings of the 1st conference on First Workshop on Hot Topics in Understanding Botnets (USENIX Association Berkeley, CA, USA, April 2007), available in electronic format at http://www.usenix.org/event/hotbots07/tech/full_papers/provos/provos.pdf.*

[24] *Whether reasonable or not, users are not willing to wait more than a couple of seconds to ascertain whether they can open, execute or view a file, or safely browse a website; nor are they willing to put in the time or effort to gain reasonable safety proficiency to operate what is increasingly complex hardware and software.*

[25] *By way of addendum, an investigation by Associated Press has found a number of people in the USA where a third party has caused abusive images of children to be downloaded on to their computer, which often results in criminal charges that might or might not be withdrawn: Jordan Robertson, 'AP IMPACT: Framed for child porn — by a PC virus', AP Technology 8 November 2009 at http://tech.yahoo.com/news/ap/20091108/ap_on_hi_te/us_tec_a_virus_framed_me.*