



X-morphic exploitation.

Gunter Ollmann

Contents

- 2 *Executive summary***
- 3 *Morphing code***
- 9 *Obfuscation techniques***
- 15 *Malicious content delivery***
- 22 *Conclusions***

Executive summary

Browsing the Internet has become an increasingly risky business in recent years. The massive increase in vulnerabilities that can be exploited via the ubiquitous Web browser has meant that attackers have steadily adopted this vector as a primary infection route for malware payloads.

Traditionally, Web browser attacks have relied on fairly simple exploit code, typically written as scripts within HTML documents. Consequently, Web browser exploits are easy to block. Using standard regular-expression and heuristic-based signature engines, exploit patterns are easily identified, and the attack can be thwarted over the network or at the host.

To overcome such protection mechanisms, attackers adopted numerous obfuscation techniques to disguise their raw exploit code. Their methods worked well, and newer, more sophisticated obfuscation methods were developed, almost guaranteeing that signature-based engines would not be able to protect against newer threats. In a world dominated by copy-paste exploit cloning, vendors of signature-based protection systems then focused on detecting the obfuscated exploit variant and were therefore able to provide protection to their customers. Although not zero-day protection, it was sufficient for many enterprise customers to mitigate widespread infection.

The obvious attacker response is to dynamically alter the obfuscated exploit each time a potential victim visits the malicious page, effectively creating a unique exploit with each request and making it impossible for signature-based

protection engines to uniquely detect each attack instance. In the malware world, the technique of altering a malicious payload with each iteration to defeat detection systems is commonly referred to as oligomorphic, polymorphic or metamorphic manipulation.

Unlike self-replicating malware, which must carry with it the means of altering itself, Web exploit developers can host their morphing algorithms and code on the Web server itself and do not need to make that code visible to the victim. Consequently, unlike morphing malware, morphed Web browser exploits do not contain superfluous morphing code, which makes these attacks considerably more difficult to detect.

Welcome to the world of personalized, one-of-a-kind Web browser exploits and the dawn of x-morphic exploitation.

Morphing code

Decades of malware development have been a source of innovation for today's Web browser exploit developers. Advanced malware techniques designed to bypass regular-expression and heuristic-based signature engines have been maturing for many years, and the lessons learned are now being applied to Web browser exploit development.

Malware authors have researched avenues that encode, encrypt and obfuscate their own morphing code. However, the anti-virus (AV) vendors' defense strategy has been to identify the code within the malware that alters and morphs

into the next iteration. From there, AV vendors provide protection updates that extend their products' ability to detect the presence of the particular malcode. The AV vendors' strategy requires skilled malcode authors to invest evermore time into their morphing code, making it more sophisticated and larger. But AV vendors may still rapidly respond to such attack measures. Hence, the malware space exhibits a continued trend toward low-threshold targeted attacks.

In the world of Web browser exploitation, attackers can use the techniques and philosophies developed by malcode authors and reapply them without the risk of protection vendors identifying the code used to generate the custom-morphed attack.

Malware morphing

Malware authors and anti-virus researchers have developed a particular nomenclature that is applied to the methods used to obfuscate and hide malware code with each infection. The most common morphing classes found in malware development include the following:

- *Oligomorphic*—In its simplest form, the malware author ships multiple decrypt engines (or decryptor patterns) instead of just one. The malware randomly selects or builds an engine from several predefined alternatives with each malware iteration. One of the first known malware incidents to use this technique was called *Whale* (August 1990). *Whale* included several dozen decryptors, each slightly different, of which the malware randomly picked one with each file it infected.

- *Polymorphic*—An evolutionary step from oligomorphic techniques, polymorphic malware can mutate its decryptors through a dynamic build process and may incorporate “noise” instructions, for example, a No Operation instruction, or an instruction to load an unused register with an arbitrary value along with randomly generated or variable keys to encrypt the constant part of the malware. This results in millions of possible permutations of the decryptor.
- *Metamorphic*—Moving beyond polymorphic techniques, metamorphic malware mutates the appearance of the malcode body. By carrying a copy of the malware source code, whenever metamorphic malware finds a compiler, it recompiles itself after adding or removing junk code to its source. Consequently, each propagated iteration of the malware will look completely different from previous versions.

Exploit morphing

Web browser exploitation suffers many of the same problems encountered by malware authors, with the probability of success being inversely proportional to the availability and subsequent popularity of the exploit code. That is, the more widely deployed and consistent the exploit code, the earlier protection is developed and deployed. Although patch availability and the ability to draw a potential victim to a malicious Web site may be limiting factors to the success of Web browser exploitation, the morphing of exploit code is a great equalizer.

Although attackers already have an armory of tools they can use to obfuscate exploit material, in the past, these techniques were either rarely applied or applied with very little consideration as to how security vendors would detect

them. As new Internet-based revenue opportunities have appeared and matured, organized criminals have begun to invest in exploit delivery platforms that can operate for longer periods of time.

The uniqueness principles of oligomorphic, polymorphic and metamorphic malware development are easily applied to commercial exploit development and are easily incorporated within Web browser attacks because of their susceptibility to content-level manipulation. Each of these malware morphing techniques may be used to dynamically obfuscate the exploit and its payload—hence the name “x-morphic” exploitation.

Web browser exploit developers have a major advantage over malware authors in the fact that the morphing code is never passed through to the victim host. Therefore no opportunity exists for the protection vendor to shortcut exploit identification by simply spotting or dealing with the x-morphic engine.

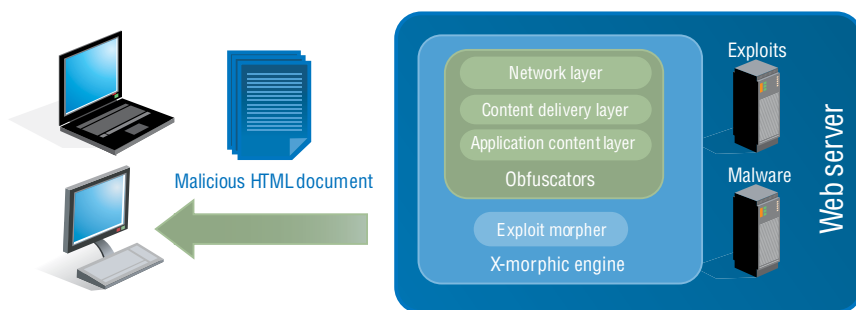
Web server delivery

The nature of Web browser vulnerability exploitation means that the victim must request content from a malicious Web server. For a Web server to be classed as malicious, it must respond to the victim’s request—an HTTP GET or HTTP POST request—with an HTTP response page that contains exploit code.

In its simplest form, a malicious Web server will serve a single HTTP page containing the exploit material with each request from a potential victim. When a signature is available for detecting that particular exploit being served, potential

victims employing this protection engine are protected. In addition, given the prevalence of security-related Web crawlers and search engines, the longer the malicious Web server continues to serve the same exploit material, the higher the probability it will be noticed and shut down by a security vendor. In such a case, the attacker not only loses control of the delivery platform but also exposes himself to probable prosecution.

Attackers are developing a solution that employs what amounts to an x-morphic “engine” designed to serve highly obfuscated and one-of-a-kind exploits with each page served to a potential victim. The x-morphic engine sophistication may include a malicious server-side script posted to a legitimate Web site or a custom Web server engine incorporating all obfuscation and morphing technologies and built into a stand-alone service that could be deployed as part of a standard botnet agent.



The concepts and mechanics behind an x-morphic engine are relatively simple, with the individual techniques and technologies having been deployed in the wild for many years. However, organized criminals have driven the demand for a simpler and more reliable delivery platform. In essence, an x-morphic engine consists of two core elements:

- *Exploit morpher: The exploit morpher element focuses on manipulating a stock Web browser exploit—for example, MS05-013-Microsoft MS-ITS CHM file code execution—by reordering, padding, swapping shellcode, changing script components and otherwise randomly altering the exploit code using oligomorphic and polymorphic principles initially perfected by malware authors.*
- *Obfuscators: The obfuscator element consists of engines working at the network layer, content delivery layer or application content layer that take the morphed exploit code and wrap it in one or more layers of obfuscation. Each obfuscation layer can add its own randomness, thereby providing a metamorphic aspect to the final exploit.*

In practice, there is no limit to the number of mutation variations. In addition, the attacker is not limited to a single exploit being contained within the malicious server response. Attackers already serve malicious pages that contain multiple exploits and use client-side script elements to logically cycle through various exploits on the victim's host until it finds one that works.

Consequently, the x-morphic engine may also include additional exploits stored on the Web server (perhaps in a database for easy use) as well as use various custom malware payloads.

Of course, there is no reason that x-morphic engine functionality cannot be extended further by employing metamorphic techniques to build custom malware from locally stored source code for each and every page response.

By doing so, the malware could be made unique and would not have to contain its own metamorphic code, enabling the malcode author to keep the morphing technology secret from anti-malware authors.

Obfuscation techniques

The developer of an x-morphic engine has a multitude of obfuscation techniques available. When incorporated into an automatic attack delivery platform, these techniques can be categorized as:

- *Network layer—fragmentation, etc.*
- *Content delivery layer—Base64 encoding, etc.*
- *Application content layer—scripting, etc.*

These obfuscation techniques are known by the security community—both professional and underground—and are commonly deployed in many security assessment and hacking tools. They each have varying levels of success against today's protection technologies. However, when combined, they create a complex threat that is difficult to protect against.

Network layer

Because the attacker has a high degree of control—if not full control—of the host serving the x-morphic exploit material, a number of techniques can be applied to the network layer to obfuscate the attack. The intent of obfuscating at the network layer is typically to bypass networkcentric protection systems such as firewalls, intrusion detection systems (IDSs), intrusion protection systems (IPSs) and filtering proxies.

The primary tool available to attackers at the network layer is packet fragmentation. It works by breaking down the packets that contain the malicious payload into smaller packets and mixing up the way the fragmented data is sent to, and interpreted by, the victim host.

Consider the following common techniques:

- *Simple fragmentation: The malicious payload is sent in ordered fragments—not necessarily the same size—and reassembled by the recipient.*



- *Out of sequence packets: The malicious payload is sent out of sequence and assembled correctly by the recipient.*



- *Overlapping packets: The malicious payload is broken into fragments that repeat part or parts of the payload and is sent with instructions to where the correct assembly points are so they can be reassembled by the recipient.*



- *Overwriting redundant packets: The malicious payload is broken down into fragments, and extra packets that contain redundant information are included. These extra fragments will not form part of the reassembled payload.*



- *Packet timeout: The malicious payload is broken into fragments and sent slowly to the recipient. The objective is to pause between each fragmented packet just long enough so that any intermediary protection devices will time out while watching the traffic and discard previous packets – ultimately having no ability to reassemble the entire packet.*



Content delivery layer

The primary delivery vehicle for Web browser exploits is HTTP. Consequently, the attacker may obfuscate the content of the exploit using any number of the many different methods of encoding data that are supported by the HTTP.

To identify exploit material within HTTP content, the protection technology must be able to not only reassemble the complete message but also correctly parse each of the encoding techniques. This requirement allows the attacker to make use of the following techniques to obfuscate and hide the malicious content:

- *Encryption over Secure Sockets Layer (SSL) and Transport Layer Security (TLS) in the form of HTTP over SSL, or HTTPS*
- *HTTP-supported compression such as “gzip” (an encoding format produced by GNU zip), “compress” (an encoding format produced by the UNIX[®] compress program) and “deflate” (the zlib encoding format)*
- *Multiple character set encoding, such as ASCII, UTF-8, UTF-7, UTF16LE, UTF16BE, UTF-32LE, UTF-32BE, etc.*
- *Transfer encoding, such as “chunked” and “token-extension”*
- *Chaffing content with characters that will not be rendered by the Web browser when encoded to a particular character set*

Some of these obfuscation techniques are best illustrated through example. Consider the Microsoft vulnerability MS04-009, which allowed an attacker to construct an HTML page that would cause Microsoft[®] Outlook to remotely start and execute code of the attacker’s choice.

To conduct the attack, the attacker would embed something similar to the following inside malicious Web page:

```

```

Consider the effect of chunked encoding on the following attack string:

<pre>Transfer-Encoding: chunked Content-Type: text/html 5 <html 9 > <body> 5 <img 4 src= 4 "mai 4 lto: 5 aa&qu 3 ot; 2 /</pre>	<pre>7 select 5 javas 5 cript 6 :alert 9 ('vulnera a ble')"> </ 8 body> </ 6 html> 1 0</pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------

Using a 7-bit Unicode encoding system, the attack string is obfuscated in the following way within a basic HTML page:

```
+ADw-html+AD4 +ADw-body+AD4+ADw-img src+AD0Alg-mailto:aa+ACY-quot; /select
javascript:alert('vulnerab le')+ACIAPg+ADw-/body+AD4 +ADw-/html+AD4
```

Finally, consider the same attack string Base64 encoded with some chaff thrown in for good measure:

```
P[G;.?h0bWΔ{#w_+%_~&%}|<Dxib!&2$R'5lPg,^o8(;aW1nl:$H );_N'-  
?>yYz$0i(\*-?bWF>p^b.&HRv}OmF#.hJn%#:F1b3Q'7_IC{9(#@z#.ZΔW}xlΔY&3Qg[amF*2YX#N^  
}|^?^ j)cm$>Δl%w,dD"$p](hb.Δ\^#GVy'>d@!!Δ-Cgnd'n[ Vsb](m'VyYWΔJsZS#c'  
!)#p'l@%j4KP'C9i~b.:2]R5{P?$i;A_8L *,2)h}0)@bWwΔ+Cgo=
```

Clearly, even with commonly available content encoding systems, the attacker has many ways to obfuscate at the content delivery layer, making visible inspection difficult. Signature-based detection and prevention systems can only function successfully if the malicious content is correctly parsed and the encoding systems are removed prior to analysis.

Application content layer

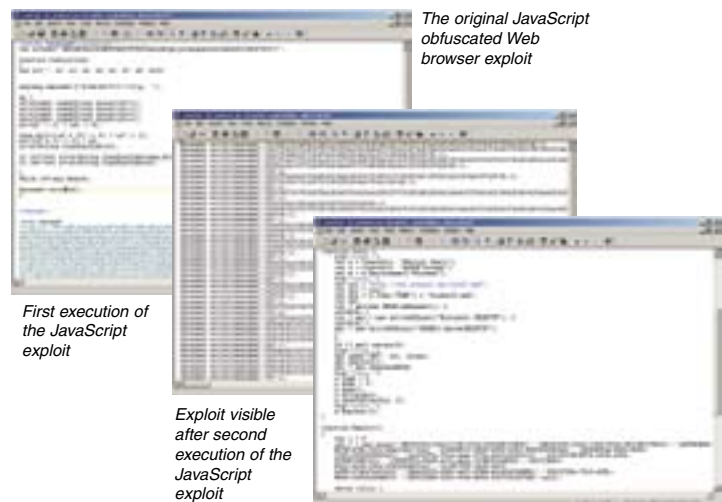
Although the content delivery layer focuses on the correct decoding of the HTML message content, further opportunities exist for the attacker to obfuscate malicious content by leveraging the way the application rebuilds, compiles or executes HTML content. Obfuscation at the application content layer is the “new frontier” for attackers.

For security products to protect against application content layer obfuscation, they must typically be able to correctly interpret or replicate much of the functionality of the Web browser application.

Some of the most popular application content layer obfuscation techniques employed by attackers include the following:

- *Splitting up the source files and dynamically rebuilding the exploit page, for instance, the use of multiple file inclusions such as .css files and .js files.*
- *Execution of embedded scripts to “unpack” and subsequently execute the exploit, often inside a new Web browser window or frame.*
- *Using supported file formats such as Flash and Adobe® Acrobat® files, which have their own scripting languages and can be rendered inside the Web browser.*

At this time, the use of JavaScript language to dynamically decode and execute an obfuscated exploit is proving popular. Consider the following example, which actually obfuscates the exploit using two levels of JavaScript encoding.



Malicious content delivery

For an attack to be successful, the attacker must cause the potential victim to request a page from the malicious Web server. The more people the attacker can entice to connect to the malicious Web server, the higher the probability of successful exploitation and subsequent installation of malware.

Some of the most popular methods attackers currently use to entice their victims include:

- *Spam*—employing e-mail, instant messenger and any other messaging platform that can deliver a message directing potential victims to the location of a malicious Web server
- *Phishing*—using the same messaging systems as spam; however, the message contains a strong aspect of social engineering (techniques used to manipulate people into performing actions or divulging confidential information, typically a personal and compelling event)
- *Hacking*—exploiting flaws in preexisting popular Web sites or Web pages that have high traffic flow and embedding links to x-morphic content
- *Banner advertising*—using banners or commercial advertising channels to create an advertisement (typically seen on most commercial Web sites) directing potential victims to a malicious Web server
- *Search page-rank*—manipulating popular page-ranking systems used by popular search engines to ensure that the malicious Web server appears high on the list of URLs returned by a search engine when potential victims search for certain words and phrases
- *Expired domains*—purchasing the expired domain registration names for popular and well-visited sites (many fail to renew their domain registrations on time) and associating the entire domain (and all associated host names) to the Internet Protocol (IP) address of a malicious Web server

- *Domain Name Server (DNS) hijacking—manipulating DNS entries on poorly secured DNS servers and altering them to direct potential victims to a malicious Web server (similar to expired domains)*
- *Forum posting—visiting popular online forums and message boards and leaving messages containing URLs to a malicious Web server.*

Attackers may also leverage exploited systems to help deliver their malicious content. For instance, attackers have already used the following methods to deliver custom Web browser exploit material:

- *Tickers and counters—Many Web pages include or reference scripts held on other Web sites to provide interesting elements to page content. A common shared component includes scrolling tickers and hit counters. In the past, attackers have compromised Web servers that provide this shared content and appended their malicious exploit material to the served content, allowing them to massively increase their potential victim audience.*
- *404 page errors—When a Web browser requests a page that doesn't exist, the Web server is designed to serve an error page to the requester. By editing the generic error page to include their exploit material, hackers can not only reach a lot of potential victims but also unexpectedly hide their content from regular users. In previous attacks, attackers have used spam e-mail to draw potential victims to nonexistent URLs on a previously compromised (but legitimate) Web server, which resulted in a maliciously encoded error page being returned from the server. After successful exploitation, the server redirects victims to the legitimate page. In this fashion, victims may never know that they are infected.*

- *Server-side user-agent checks – With each page request, a client Web browser will send information about itself, including elements such as the referrer page and the Web browser type. Attackers are already leveraging this information to ensure that exploit code is only served to pages most likely to be vulnerable to it (using the user-agent information) and are using referrer information to decide whether their potential victim arrived from a linking site they set up (i.e., if the referrer information is incorrect, attackers may assume that a security researcher is searching for their malicious content and choose not to serve up the malicious content).*

It is important to realize that while attackers can morph both the Web browser exploit and malware payload using an x-morphic engine, they can also tune their delivery to a specific visitor. This is considered a personalized attack rather than a targeted attack.

In a personalized attack, although they have very little information about the user, attackers seek to tune their exploit delivery specifically to their potential victim's Web browser and system – trying to provide the exploit with the highest level of success based on the information it gained from the Web browser page request as well as ensuring that any future discovery of the x-morphic engine is difficult (if not impossible).

Strategies that the x-morphic engine developers will likely adopt as part of their personalized attack delivery platform include the following:

- *Using the source IP address information of the request, the attacker can ensure that only one exploit is ever served to that address. This helps prevent subsequent replay-based analysis. For example, a potential victim visits a Web server and a malicious page is served. The victim noticed that the Web browser stuttered, crashed or restarted and tries to request the Web page again to view the page source and find out what happened.*
- *Attackers may choose to implement a time-based approach to protect their engine from discovery. For example, the engine may only operate for ten minutes every four hours. Unless security researchers examine Web server content at the right time, they will never see the x-morphic engine working.*
- *By observing the specific browser-type information, the attacker would ensure that only exploits relevant to that particular browser are ever served. This would mean that search engines and Web crawlers would never be exposed to the malicious content and could not be used for discovery. In addition, the x-morphic engine would only respond to Web browsers if it knows it has exploit code that has a high probability of successful compromise.*

- *Leveraging the IP address information, the attacker can prevent certain IP addresses or ranges from ever being served malicious content. For example, the IP address ranges of most security companies are well known. So too are those of certain geographical regions. In a reverse scenario, the attacker could choose to target only certain IP addresses or ranges.*
- *One-time URLs have been popular within spam messages as a way of validating the existence of a specific e-mail address. One-time URLs will likely be used to ensure that exploit code is only served once to anyone visiting the specific URL—making it difficult for security researchers to obtain copies of the malicious Web server’s content.*

To date, we have already observed attackers using techniques that prevent multiple requests from the same IP address on Web sites that have hosted zero-day exploit material.

Commercial endeavors

The commercialization of Internet-based crime will continue to drive the evolution of x-morphic exploitation technologies and corresponding engine development. The first generation of commercial services, for example, managed exploit providers such as Inet-Lux, have already appeared. They cater to criminal and gray-legal organizations and have been widely adopted by spyware and adware vendors.

Meanwhile, the commercialization and subsequent legitimization of vulnerability purchases has similarly driven a market for exploit purchase. Not only are zero-day exploits being purchased for high monetary value, but exploits for recently patched vulnerabilities such as those covered in a previous month’s release are also highly valued. The financial value of these exploits drops with

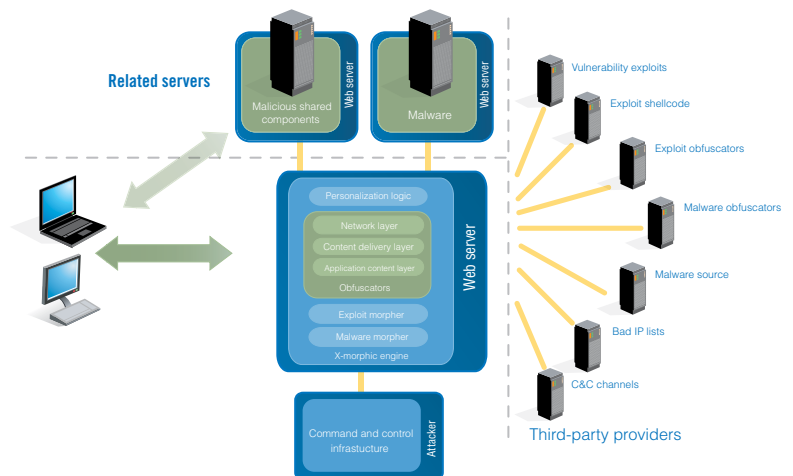
each week following the patches' original release. However, even exploits for Web browser vulnerabilities that are several months old are still of some value and can be employed with a fair prospect of successful compromise.

To date, there are several legitimate security organizations that provide commercial-grade exploitation platforms used by consultants to test the security of an organization. Freeware modular exploit tools such as Metasploit have already proved popular among attackers and malware developers, and it is anticipated that this kind of platform development will become increasingly competitive as more attackers begin to use x-morphic engines and delivery systems.

Future

For organizations that rely on malware installation as part of their business, Web browser exploit platforms will be vital for their continued infection success. X-morphic exploitation is expected to become the default method of Web-based delivery and will replace the more ad hoc exploit delivery channels (i.e., largely uncoordinated manual efforts) currently used by criminal organizations.

X-morphic engines will become more extensible, with third-party developers eventually providing specialized content that can be dropped in—likely following a subscription model. These third-parties will most likely focus first on delivering content such as exploits, obfuscation modules, malware source code and modules (e.g., malware rootkit technologies, command and control modules, etc.).



The probability of off-the-shelf x-morphic engines becoming publicly available is high and likely to be driven by penetration testing requirements. In addition, there are several exploit and attack-based open source projects (such as Metasploit) that will provide some degree of modular content to the engines – intentionally or not.

Over the next few years, we can expect x-morphic exploitation techniques to be adopted for or adapted to other server protocols beyond HTTP. Internet protocols that make use of some form of URL or are responsive to “go here for more info” messages will likely be susceptible – assuming that actual exploitation of the user’s application is a popular and successful source for vulnerabilities.

Conclusions

The successful use of Web browser exploits as a method of installing malware has driven the development of new attack technologies and the subsequent emergence of a new threat category. Lessons learned from 30 years of malware development are only now being applied in exploit development.

X-morphic exploitation and its delivery engines represent the evolutionary convergence of several attack methodologies and obfuscation techniques rather than an out-of-the-blue advance. Pressure from security vendors and operating system developers have ensured that attackers must adopt multiple layers of obfuscation to bypass point protection technologies, while automated discovery systems have forced attackers to be more careful in hiding their business assets, for example, compromised hosts, zero-day exploits, etc.

Such convergence was inevitable. However, the simultaneous growth of a commercial exploit development market and commercial hacking tools has increased the diversity of x-morphic permutation possibilities and will have a dramatic effect on protection technologies going forward. Complex international law issues have further muddied the waters to create legal gray areas that will allow new businesses to flourish, providing drop-in modules for x-morphic attack engines.

To combat this class of threat, businesses will need to adopt strong defense-in-depth protection technologies that work seamlessly together and strive to overcome the “good enough” security mentality that has been pervasive for the last half decade.

For more information

For additional information, contact your IBM sales representative or your IBM Business Partner or visit:

ibm.com/us/iss



Additional reading

- 1 IBM Internet Security Systems X-Force Newsletter, "Obfuscated Attacks," Robert Freeman, December 2006, http://www.iss.net/documents/literature/X-ForceNews_Dec06.pdf.
- 2 IBM Internet Security Systems X-Force Newsletter, "Trends in Web Exploitation," Robert Freeman, November 2006, http://www.iss.net/documents/literature/X-ForceNews_Nov06.pdf.
- 3 Security Focus, "Detecting Complex Viruses," Peter Ferrie and Frederic Perriot, December 2004, <http://www.securityfocus.com/infocus/1813>.
- 4 Addison Wesley Professional, "Advanced Code Evolution Techniques and Computer Virus Generator Kits," P. Szor, 2005.

© Copyright IBM Corporation 2007

IBM Global Services
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
05-07
All Rights Reserved

IBM and the IBM logo are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Microsoft is trademark of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc., in the United States, other countries, or both.

Other company, product and service names may be trademarks or service marks of others.

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

IBM assumes no responsibility regarding the accuracy of the information provided herein and use of such information is at the recipient's own risk. Information herein may be changed or updated without notice. IBM may also make improvements and/or changes in the products and/or the programs described herein at any time without notice.