WHITE PAPER
Security Solutions

Escaping from Microsoft's Protected Mode Internet Explorer

Evaluating a potential security boundary

Abstract

The level of protection offered by Protected Mode Internet Explorer® is not well understood and there are common misconceptions about its status as a security feature. This research set out to discover the full extent of how Protected Mode can protect users from zero-day memory corruption vulnerabilities in Internet Explorer and third-party extensions. As a result of this research, a bypass of the feature was discovered along with a number of generic attack patterns which must be protected against to prevent future circumvention of the feature.

Introduction

In Internet Explorer 7 and Windows Vista, Microsoft introduced a new browser security feature called "Protected Mode". According to Microsoft, this mechanism "significantly reduces the ability of an attack [against Internet Explorer] to write, alter or destroy data on the user's machine".[1,2] A clearer description is that the feature attempts to protect the integrity of the client machine in the event the browser is compromised in an attack and prevent malware from being persisted on the targeted machine.

This paper will describe why this is not currently the case in Internet Explorer 7 or 8 for remote code execution vulnerabilities, discuss the limitations of the feature by design, identify generic attacks patterns that can be used to bypass the feature (without user intervention) and discuss some inconsistencies in the underlying access control implemented in Microsoft® Windows®.

The Microsoft Security Response Centre (MSRC) does not regard Protected Mode as a security boundary,[3] but there is the intention for it to become a true security boundary in a future version of Internet Explorer.[4] Once it becomes a formal security boundary, Microsoft will patch any successful bypass of the mechanism within their monthly security updates.[4] However, since the feature's introduction, a wide range of sources at Microsoft and elsewhere have implied or stated security claims about the feature, for example:

- "The idea behind Protected Mode IE is that even if an attacker somehow defeated every defense mechanism and gained control of the IE process and got it to run some arbitrary code that code would be severely limited in what it could do." – IEBlog (2006)[5]
- "What's interesting about this is the fact that Firefox doesn't have the benefit of Protected Mode under Vista, which can somewhat mitigate the damage that can be done if Internet Explorer 7 is exploited by [the ANI] vulnerability." – ZDNet (2007)[6]
- "IE7 users on Vista also benefit from Protected Mode, which helps prevent the installation of malicious software, even in the event that an exploit results in code execution." – IEBlog (2009)[7]

These claims cannot carry any serious weight without Microsoft regarding the feature as implementing a security boundary. Later in this paper, a generic bypass of the feature is described which invalidates the above claims about the ability of the feature to protect against memory corruption vulnerabilities. As far as I am aware, Microsoft has publicly stated only once that Protected Mode is not a security boundary.[3]

Confusion about the protection offered by Protected Mode even exists amongst security researchers, for example:

- "Windows Vista introduced Protected Mode Internet Explorer which was a step in the right direction. An exploit against Internet Explorer on Vista and 7 will run with Low Integrity, so it can't change or harm your system. It can only upload all of your sensitive information to the attacker (Phew!)." – Dino Dai Zovi (2010)[8]
- "Likewise, Internet Explorer 7, when running on Windows Vista, is sandboxed on OS level as a whole" – Michael Zalewski (a.k.a. lcamtuf) comparing Protected Mode with the "Chrome Security Sandbox" in the 'Browser Security Handbook'.[9]

Whether or not Protected Mode or User Account Control (UAC) should be considered as security boundaries has been a topic of debate, but attention has mainly focused on the later of these two mechanisms.[10,11,12] I am only aware of a single publication, not written by a Microsoft employee, which discusses the security of the feature post-release.[13] Therefore, this paper will not consider limitations of UAC and will focus on Protected Mode and Mandatory Integrity Control (MIC).

Both Protected Mode and User Access Control are built upon Mandatory Integrity Control and both provide elevation routes between different integrity levels. As a result, both need to be understood in order to properly assess the security benefit of using Protected Mode Internet Explorer.

The rest of this paper is divided into five sections:

1. Design and implementation of Mandatory Integrity Control
2. Design and implementation of Protected Mode Internet Explorer
3. Generic attack patterns against Protected Mode Internet Explorer
4. Bypassing Protected Mode Internet Explorer
5. Conclusions and Recommendations

Sections 1 and 2 will lay the groundwork for the reader to understand sections 3 and 4.

It is hoped that this paper will accurately inform the reader of the true benefits of using Protected Mode Internet Explorer, both now and in future, when the feature is upgraded to the status of a formal security boundary by Microsoft.

1. Design and implementation of Mandatory Integrity Control

Mandatory Integrity Control (MIC) is a form of mandatory access control introduced in Windows Vista. It is an access control policy under the control of the operating system and not the user which allows the concept of a less-trusted process to be introduced. This mandatory access control complements the discretionary access control policy defined by the owners of objects which is based on users and groups.

In this mandatory access control scheme, each securable object[14] (e.g. processes, files and shared sections), has an access control entry (ACE) in the System Access Control List (SACL). This ACE is referred to as the 'mandatory label' (SYSTEM_MANDATORY_LABEL_ACE) and there is only a single ACE of this type, any surplus mandatory labels are ignored and leave the Access Control List in a non-canonical form.[15]

The Security Identifier (SID) specified in this access control entry represents the level of "trustworthiness" (or integrity level) of the object which the SACL belongs to and the 'mandatory label rights' defines one or more MIC policies which apply to the object. These SIDs are of the form:

S-1-16-*, where '*' is the value represents the integrity level represented by the SID.

Before any discretionary access control is performed (based on users and groups), the mandatory access control check is done using the integrity level of the requesting process (derived from its primary token) and the mandatory label of the object being accessed. If this check passes, then the discretionary access check can proceed.

The most common policy is "No Write Up" which prevents processes at lower integrity levels from writing to objects at higher integrity levels and gives the feature its more common name of "Integrity Levels". This prevents a less-trusted process within a user's session from corrupting more trusted objects within the same user session, even when the discretionary access control policy would allow it. In this way, Mandatory Integrity Control introduces the concept of less-trusted applications running under a single user account.

There are two other policies defined; "No Read Up" and "No Execute Up".

The "No Read Up" policy prevents read operations from lower integrity processes and provides confidentiality. This is used, for example, to prevent the reading of the virtual address space of a higher integrity process.

The "No Execute Up" policy is used by DCOM to control the launch and invocation of out-of-process COM objects, which could run at a higher integrity level to the COM client. If the DCOM subsystem was not integrity aware, then higher integrity COM objects could be made to performed actions on behalf of a lower integrity process.[16]

There is not a default policy because each type of securable object has a different set of policies that apply to it. For example, processes are marked as both No Write Up and No Read Up. This prevents a lower integrity process from reading or writing the virtual memory space of a higher integrity process.

There are a total of six integrity levels explicitly defined by Windows: Un-trusted, Low, Medium, High, System and Protected (which is used for implementing Digital Right Management). But there are actually 216 integrity levels as this value is represented by a 16 bit number. The starting integrity level for a logon session varies according to the user:

Integrity Level	User
System	Local System NT Services
High	Elevated Administrator
Medium	Un-elevated Administrator Limited Users

When a new process is launched, the child process inherits the integrity level of the parent process or the parent process can launch the child process with a lower integrity level, but not a higher one.

When a process accesses an object, including another process, the integrity level of the requestor has to dominate (be greater than or equal to) the integrity level of the resource; if it does not, then the requested access is checked to see if it conflicts with the MIC policy on the resource. For example, if the No Read Up policy is specified then read requests will be rejected.

There are two main types of access rights, generic access rights (read, write and execute) and object-specific access rights (e.g. terminate a process, read a file, or add a file to a directory). Since MIC policies are defined in terms of generic access rights, the mapping to object-specific access rights is important, as they determine precisely which operations are disallowed by the policy. Once the full set of permitted operations on higher integrity objects is known, we can look for operations which may facilitate privilege escalation.

This mapping between object-specific access rights and generic read, write and execute permissions is defined per resource type and stored in a GENERIC_MAPPING structure. The structure is passed to access control functions in the Windows API and is defined below:[17,18]

```
typedef struct _GENERIC_MAPPING {
    ACCESS_MASK GenericRead;
    ACCESS_MASK GenericWrite;
    ACCESS_MASK GenericExecute;
    ACCESS_MASK GenericAll;
} GENERIC_MAPPING;
typedef GENERIC_MAPPING *PGENERIC_MAPPING;
```

As an example, generic write access to a file maps to the right to modify or append to a file's contents and change a file's attributes on the file system.

Specific mappings which were deemed to be of interest are listed below:

- The PROCESS_TERMINATE access right is granted on a higher integrity processes, but THREAD_TERMINATE is not. This allows any process in the same session to be terminated by a less trusted process. If the security of a system relies on a process to be running at all times, then granting this access right may introduce a security issue.
- The KEY_CREATE_LINK access right allows registry soft-links to which point to any key within the same registry hive, regardless of the integrity level associated with the target key. HKEY_CURRENT_USER is one example of a registry soft link.[19] This may allow a medium integrity process to be tricked into modifying a medium integrity registry key. Also, a number of vulnerabilities have recently been discovered in this functionality which could be exploited by low integrity processes.

- The GENERIC_WRITE access rights are granted on a higher integrity Mutant (Mutex), but not on a Semaphore or an Event. Mutants, Semaphores and Events are all synchronisation objects.
- Requesting no access on a higher integrity file is permitted, but this is not true of other types of objects. This is merely an inconsistency, which may or may not have any security impact.

These GENERIC_MAPPINGs may have been influenced by a number of factors including application compatibility. But now that they have been chosen, they are unlikely to change and therefore define potential avenues of attack when trying to elevate between integrity levels.

Until now, only securable kernel objects have been discussed, but there are a number of other resources which are not subject to Mandatory Integrity Control. One example is network sockets. This resulted in the first bypass of Protected Mode Internet Explorer. This issue was fixed in Windows Beta 2.[20]

Mandatory Integrity Control does not cover a number of other resources including the Service Control Manager; this would allow weak service access control to be exploited, even by a low integrity process.

The most important limitation of Mandatory Integrity Control is that it only provides integrity of the operating system. As a result, compromise of a low integrity process will not protect the confidentiality of a user's files. But by protecting the integrity of a user's data, recoverability can be provided; due to the fact that Protected Mode cannot be bypassed then Malware will be unable to persist across reboots. A side-effect is that if you do not ever reboot your workstation, Malware will be able to persist for long periods of time.

2. Design and implementation of Protected Mode Internet Explorer

To protect the integrity of the user's machine in the case of successful exploitation, un-trusted web sites are rendered in low-integrity Internet Explorer tabs. In Internet Explorer 8, these tabs sit side by side with other tabs rendered at other integrity levels though the Loosely-Coupled Internet Explorer functionality (LCIE), whereas in Internet Explorer 7 these tabs would have resided in different browser windows entirely.[21] In both cases, an Internet Explorer broker process coordinates tabs and windows both inside and outside of Protected Mode.

The broker process runs at either medium (un-elevated) or high integrity (elevated) and provides functionality to the low integrity browser windows via the Protected Mode API.[22] Communication is implemented via Local Procedure Calls (LPC).

The Protected Mode API is very restricted and many of the actions require user intervention to succeed, therefore this API has not been extensively investigated as part of this research. There is another interface between the broker and low integrity processes which is implemented using a shared section and mutex called "LRIEElevationPolicy_" and "LRIEElevationPolicyMutex" respectively, but again this has not been investigated.

An application compatibility shim layer (IEShims.dll), which intercepts function calls (by inserting hooks) in the low integrity process, allows the Internet Explorer process to transparently run at low or medium integrity by rewriting file system and registry paths and redirecting process launch requests to the broker process. The following APIs and their variants are hooked:

- CreateProcess()
- CoCreateInstance()
- CoGetClassObject()
- File access functions
- File path functions
- Registry access functions
- GetProcAddress()
- LoadLibrary()

ShellExecute() is not hooked by the shim, but it may call CreateProcess(), which is hooked.

Calling the CreateProcess, CoCreateInstance or ShellExecute will transfer the process launch request to the higher integrity Internet Explorer broker process. The Internet Explorer broker process will then make a decision on the application launch as per the policy stored in the system registry at:

(HKLM|HKCU)\SOFTWARE\Microsoft\Internet Explorer\Low Rights\ElevationPolicy\[23]

Quoting from Microsoft's documentation, the policy options for how Protected Mode should launch a registered broker are:

Value	Result	Example
3	Protected Mode silently launches the broker as a medium integrity process.	winword.exe
2	Protected Mode prompts the user for permission to launch the process. If permission is granted, the process is launched as a medium integrity process.	All other processes (default setting)
1	Protected mode silently launches the broker as a low integrity process.	iexplore.exe
0	Protected mode prevents the process from launching.	cmd.exe

Through the hooking of the low integrity Internet Explorer process, the Protected Mode API exposed by the Internet Explorer broker process and other application compatibility techniques, a large number of in-process Internet Explorer extension work in low integrity without modification. However, other more complicated add-ins and applications require modification. As a result of this incompatibility and Microsoft's dedication to backwards compatibility, not all Internet Explorer zones render their member sites in Protected Mode. Each Internet Explorer zone defines a set of security policies for pages rendered in that zone and enabling Protected Mode is one of the available settings.

In the latest versions of Internet Explorer 7 and 8, the default policies for workstations are:

Zone	Internet Explorer Policy
Internet	On
Local Intranet*	Off**
Trusted Sites	Off
Restricted Sites	On
Local Computer	Off

* The Local Intranet Zone is disabled unless the workstation is domain joined, or the user explicitly enables the zone when prompted.[24]

** Protected Mode was previously enabled for the Local Intranet Zone.[25]

This policy means that whether Protected Mode is enabled or not is predicated on the zone membership of a page being loaded, which has significant implications for the feature.

3. Generic Attack Patterns against Protected Mode Internet Explorer

In this section, a number of attack patterns are described which either individually or combined, could allow the protection offered by Protected Mode to be bypassed in different circumstances. The section focuses on attacks which do not have standalone value such as local kernel exploits or which become more significant when attacking Protected Mode Internet Explorer.

The first pattern is that of a remote Internet Explorer Zone escalation. In this attack, an attacker has a web page rendered in one zone, normally in the "Internet Zone". From this webpage they are able to get malicious content rendered in a more permissive zone, where Protected Mode is disabled, such as the "Local Intranet Zone" or the "Trusted Sites Zone". A number of ways to do this are possible including:

- Spoofing a website in the Trusted Sites List which is accessed over HTTP instead of HTTPS.
- Having a web server address which is reachable through an address which is recognised as a member of the Local Intranet Zone (e.g. UNC paths, unqualified hostnames).[26]
- Persistent or Reflective Cross Site Scripting (XSS) attacks against sites rendered in an Internet Explorer zone where Protected Mode is disabled.

Also, any URL parsing bugs in the implementation of IInternetSecurityManager::MapUrlToZone() would also allow Internet Explorer Zone escalation.[27]

A local Internet Explorer zone escalation attack is the same as a remote Internet Explorer zone escalation attack except the attacker is already able to execute arbitrary code on the same machine as the victim.

This code could be executing at low integrity as the victim, or as another user at medium integrity.

Since Mandatory Integrity Control does not apply to network sockets, a connection could be made over the loopback interface to a local network service and the server will not know the integrity level of the client process and thus will not be able to incorporate that information into an authorisation decision. Therefore the service may be lured into performing a privileged action on behalf of the client. Furthermore, services may be listening on the loopback interface because they expose functionality which should not be exposed to un-trusted networks. It was through connecting to an SMB share as an Administrator that first allowed a bypass of Protected Mode, but there are other services which are exposed over the loopback interface including the "Simple Service Discovery Protocol" which implements Universal Plug and Play support. Also, a number of third-party services are also bound to the local interface including the Intel "User Notification Service".

The next pattern is the use of local exploits that explicitly target Internet Explorer's trusted brokers (those which can silently elevate). Trusted Brokers can be attacked in malicious command line arguments which results in either the broker performing a privileged operation on behalf of the low integrity process or the execution of arbitrary code through memory corruption vulnerabilities.

Additionally, any other medium and high integrity applications in the user's session can be attacked through a wide range of vectors.

One vector is through name squatting attacks in the user's "BaseNamedObjects" (BNO) kernel object namespace. In this attack, an object with a fixed name can be created which is then opened by an application that trusts the object not to be malicious by virtue of it existing in the local namespace (which was previously a reasonable assumption). This issue has been given as an example of why Protected Mode is not a security boundary by Microsoft.

Another vector is through leaked or duplicated handles. Access control decisions are made at the point that an object is opened, so existing handles may provide access to resources that are only accessible to more privileged contexts if they are transferred between processes. Handles in low integrity processes which have write access rights to higher integrity objects can be considered privileged. It was through this vector that Skywing escaped from Protected Mode using a leaked handle.[13]

The last vector is through objects which are deliberately shared between low integrity processes and higher integrity processes. This includes the Window Station kernel object which is shared between all processes within the same interactive logon session. With full access to the Window Station, low integrity processes also have access to the Global Atom Table, Window Station properties, the user's clipboard and the "\Default" Desktop object. Such objects can be detected through a tool written as part of this research that locates objects open in low and higher integrity processes; to determine if two handles refer to the same object, the kernel mode pointers to the object's data structure are compared.

The Global Atom Table is used to store both integers and strings which are each indexed by an integer. A simple fuzzer was created to fuzz this table, which only caused a null pointer dereference in "Process Explorer" and corruption of Internet Explorer UI elements. Dynamic Data Exchange (DDE) inter-process communication occurs through the Global Atom Table and this may be subject to more interesting attacks via malicious atom table manipulation.[28] Internet Explorer also uses the Global Atom Table heavily, but it would seem mostly for User Interface related functionality.

Through access to the clipboard, apart from sniffing the clipboard contents, it is also possible to match a string in the clipboard and replace it with another. The user would then paste the attacker controlled string into a higher integrity application. As a simple example, pasting a command line with a trailing new-line character into the command shell would result in arbitrary command execution. But a large number of other possibilities exist.

4. Bypassing Protected Mode Internet Explorer

During this research, one generic and reliable method for escalating privilege from low to medium integrity was discovered which required no user interaction. However, it does require the Local Intranet Zone to be enabled (as it is for domain joined workstations by default). The attack combines the facts that sockets are not subject to Mandatory Integrity Control and that sites in the Local Intranet Zone are rendered with Protected Mode disabled.

The attack assumes the existence of exploitable memory corruption vulnerability within Internet Explorer or an extension, which is the precise scenario that Protected Mode is supposed to mitigate.

Once the initial remote exploit has been used to execute arbitrary code at low integrity on the client, the payload can create a web server listening on any port on the loopback interface, even as a limited user at low integrity. The web server should be able to serve-up the original exploit that allowed remote exploitation in the first instance. Since the exploit will now be launched from the same machine, exploitation can be made significantly more reliable as Address Space Layout Randomisation (ASLR) is no longer effective and other exploitation techniques can be used with higher probabilities of success.

The browser can be instructed to navigate to this new malicious web server using the IELaunchUrl() function, which is callable from low integrity as part of the Protected Mode API. This will cause a new tab to be launched which will navigate to "http://localhost/exploit.html" or similar.[29]

The new malicious web page will be rendered in the Local Intranet Zone and the rendering process will now be executing at medium integrity. By exploiting the same vulnerability a second time, arbitrary code execution can now be achieved as the same user at medium integrity. This provides full access to the user's account and allows malware to be persisted on the client, something which was not possible from low integrity whilst in Protected Mode.

This attack works in both Internet Explorer 7 and 8.

The fact that a single exploit can be used for both the remote exploit and local privilege escalation is central to why this is a significant issue. Features such as Protected Mode can only be effective if they either significantly raise the cost of an attack, or reduce the probability of a successful attack. In aggregate, a combination of ASLR, GS Cookies, SafeSEH and DEP do indeed require more sophisticated exploits. As a result, often a second flaw needs to be discovered to permit reliable exploitation. But as I have shown above, a single exploit can be combined with a generic technique to allow Protected Mode to be bypassed, so the cost of exploitation has not been materially increased assuming a generic technique is known.

Bypassing DEP, ASLR, SafeSEH in Internet Explorer 7

In order to make the initial remote exploit reliable when faced with DEP, ASLR and SafeSEH in Internet Explorer 7, there is a weakness that has been introduced by Microsoft Detours. Detours implements user-mode function hooking in Internet Explorer. This is due to a readable, writeable and executable memory region at a fixed location (0x5fff0000 on 32 bit systems) created by Detours.

By overwriting this region with arbitrary code, all these exploit mitigations can be bypassed at once to allow reliable remote exploitation. Once the region of memory has been overwritten, triggering a dialog box (e.g. a "Save As..." dialog) will cause the attacker's malicious code to be executed immediately before the dialog is supposed to appear.

This issue is fixed in Internet Explorer 8 as the section is no longer marked as writeable.

5. Conclusions and Recommendations

Given the current set of potential ways to bypass Protected Mode's protection by locally escalating from low to medium integrity, it can be concluded that the mechanism currently provides little in the way of reliable protection from remote code execution attacks. However, currently, most malicious code that runs at low integrity will likely fail to persist across reboots, since it will not be aware that it is running at low integrity. For example, at the time of writing, the Metasploit Framework is generally not integrity level aware.

Furthermore, the use of integrity levels alone cannot be used to create a sandbox on the Windows Vista platform, but it may be a useful addition to the use of restricted access tokens.

As an Administrator, to help secure your systems, take the following steps:

- Ensure that UAC is enabled; as disabling it will also disable Protected Mode.
- Ensure that workstation users do not run as Administrators.
- Enable Protected Mode for all zones where possible.
- Disable the Local Intranet Zone, or limit the members of the zone as far as possible.

- Ensure that third-party software vendors create software which does not incorrectly configure Internet Explorer's elevation policy and introduce privilege escalation bugs that allow malicious code to escape from Protected Mode.
- Configure group policy to prevent users from configuring the Internet Explorer elevation policy. This will prevent users from unwittingly introducing new privilege escalation bugs.

Another more extreme mitigation would be to prevent the launch of any broker processes which are not listed in the elevation policy by setting the default action to 'deny'.

Since this research, a number of other "Protected Mode" applications have been created. Although this is a positive sign that vendors are looking for mechanisms beyond patching to protect users, these other products will undoubtedly be affected by the issues detailed in this whitepaper. But the reality is that until Microsoft considers Protected Mode Internet Explorer a security boundary, it will be difficult to gain any level of assurance in other Protected Mode implementations such as Microsoft Office and Adobe Reader.

Although Microsoft intend for a later version of Internet Explorer to introduce Protected Mode as a formal security boundary, no official plans have been released. Prior to this, third-party software vendors need to become aware of how their applications need to be secured, such that they do not invalidate any security benefits provided by Protected Mode Internet Explorer.

Finally, Microsoft and other software vendors should clearly document which features do and do not have associated security claims. Clearly stating which features make security claims, and which do not, will allow informed decisions to be made on IT security issues. This will benefit vendors, customers and even security researchers.

Further research is needed to determine what changes should be necessary to allow Protected Mode to be promoted to the status of an official security boundary and to validate the implementation of this design.

References

1 http://msdn.microsoft.com/en-us/library/bb625964.aspx
2 http://msdn.microsoft.com/en-us/library/bb250462.aspx
3 http://www.microsoft.com/emea/spotlight/sessionh.aspx?videoid=993
4 Microsoft Security Response Centre - Private correspondence.
5 http://blogs.msdn.com/b/ie/archive/2006/02/09/528963.aspx
6 http://www.zdnet.com/blog/ou/firefox-ani-exploit-on-the-way-no-protected-mode/461
7 http://blogs.msdn.com/b/ie/archive/2009/07/15/internet-explorer-s-activex-security-mitigations-in-use.aspx
8 http://threatpost.com/en_us/blogs/i-have-only-one-security-prediction-2010-010610
9 http://code.google.com/p/browsersec/wiki/Part3#Open_browser_engineering_issues
10 http://theinvisiblethings.blogspot.com/2007/02/vista-security-model-big-joke.html
11 http://www.tgdaily.com/software-features/41346-updated-microsoft-uac-isnt-broken-you-just-dont-get-it
12 http://www.istartedsomething.com/20090131/microsoft-dismisses-windows-7-uac-security-flaw-insists-by-design/
13 http://www.uninformed.org/?v=8&a=6&t=pdf
14 http://msdn.microsoft.com/en-us/library/Aa379557
15 http://msdn.microsoft.com/en-us/library/aa374928.aspx
16 http://msdn.microsoft.com/en-us/library/bb625962.aspx#sectionSection2
17 http://msdn.microsoft.com/en-us/library/aa446633.aspx
18 http://msdn.microsoft.com/en-us/library/aa374892.aspx
19 http://www.drdobbs.com/184416290
20 http://www.symantec.com/avcenter/reference/Windows_Vista_Security_Model_Analysis.pdf
21 http://blogs.msdn.com/b/ie/archive/2008/03/11/ie8-and-loosely-coupled-ie-lcie.aspx
22 http://msdn.microsoft.com/en-us/library/cc848890.aspx
23 http://msdn.microsoft.com/en-us/library/Bb250462.aspx#wpm_elebp
24 http://blogs.msdn.com/b/ie/archive/2005/12/07/501075.aspx
25 http://blogs.msdn.com/b/askie/archive/2009/02/17/protected-mode-now-disabled-for-the-local-intranet-zone-in-internet-explorer-8.aspx
26 http://support.microsoft.com/kb/303650
27 http://msdn.microsoft.com/en-us/library/ms537133(VS.85).aspx
28 http://msdn.microsoft.com/en-us/library/ms649053(VS.85).aspx
29 http://msdn.microsoft.com/en-us/library/aa767962%28VS.85%29.aspx