Malzilla

I do not like to write any kind of documentation (not to say that I hate to do it), but there are too many people asking for a bit of instructions…

So, what the heck is Malzilla?
Hmmm… hard to explain. Not that I'm so bad at explaining, but this is really a complicated animal.

1. It downloads web content, but does not render it, so it is not a web browser. You can compare this part of Malzilla with WGET, just fetching a document.

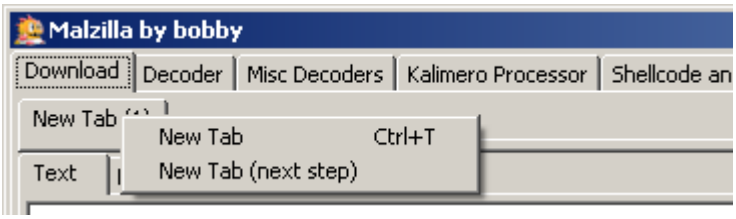
2. It can do some limited running of JavaScript, but just as much as we need to take a look to see what it does.

3. There is also a shellcode analyzer that works in emulated environment, so the shellcode does not affect our real system.

4. If our obfuscated data or shellcode does not get automatically deobfuscated etc. there is also a full bunch of tiny functions that should help us to do the job manually

Download tab

Let's explain what we have on Download tab.



Download tab can have one or more sub-tabs. You can have more than one open document at time.
If you click with right mouse button on sub-tabs, you will get a menu like seen on shot.
You have options to create new tab, close tab (can't be seen on the on the shot above) **New Tab (next step)** will create new tab, but it will take URL of the current tab to be the referrer on the next tab. Cookies are also passed to the new tab. This is useful if you want to track the links which use referrers (various affiliate links).

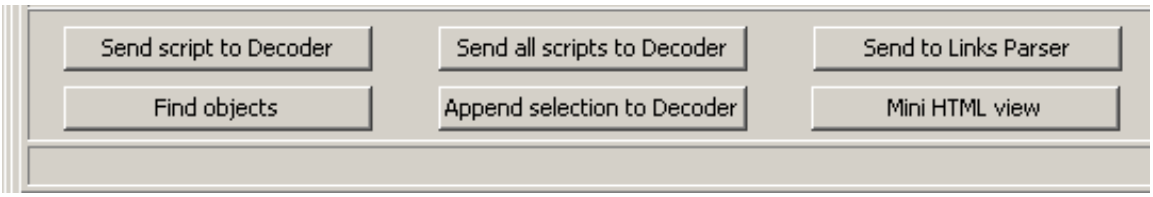
Each downloader tab contains 4 tabs.

Hex tab is hex view of the data in Text tab (same content). If you are retrieving binary data, there is no much sense to look at it as to a text, so you will probably use hex view. Additionally, on Text tab can happen that you see just partial data because on the text tab will stop reading the stream if binary zero (null) occurs in stream.

Cookies tab will contain the cookies set by the server. Just the cookies set by server are here. Cookies set through HTML or JavaScript are not processed.

Link Parser contains two sub-tabs: Links and IFrames.
Just the HREF links are parsed. Links from textual part of a HTML document are not parsed if these are not under HREF tags. Same goes for links from comments – not parsed.
Important thing to see is the box **URI base**.
If the document contains HTML tag that explicitly defines the base for relative links, the label **URI base** will be changed to **URI base (detected)**.
If the tag is missing, Malzilla will use current URL to get the URI base, and the label will be changed to **URI base (not detected)**.
More info on Base tag: http://www.w3schools.com/TAGS/tag_base.asp

Middle button panel, part one:



Send script to Decoder will search for JavaScript in the HTML document and copy/paste it to Decoder tab. Clicking on it for the first time will search for first occurrence of JavaScript in the document. Clicking it after that will search for the next JavaScript and so on. If the end of document is reached, it will search from the beginning of document again.
Holding CTRL while clicking on this button will create a new sub-tab on Decoder tab, and copy/paste the script there.

Send all scripts to Decoder will traverse the document and copy/paste all the scripts to the Decoder. Pasting to new Decoder sub-tab is also available if clicking while holding CTRL key.

Send to Links Parser will try to find all the HREF and IFrame links. Results are shown on Links Parser tab.

Find objects. As relevant data can reside outside of JavaScripts, this function will try to make a list of objects that can contain such data.
List of objects is defined in file **HTML_Obj_list.txt**
This function works pretty well for the documents where relevant data is in HTML events like in OnClick event.
The format of **HTML_Obj_list.txt** is in the form of CSV file:
`keyword::delimiter::data_begin_sign::delimiter::data_end_sign`

Example:
`onClick;";"`

This means: search for the keyword `onClick`.
Find the occurrence of quotation mark after the keyword. From there on copy the data.
Find the occurrence of quotation mark after that. Do not copy data anymore.

Append selection to Decoder – simple copy/paste of selected part of document to Decoder tab.

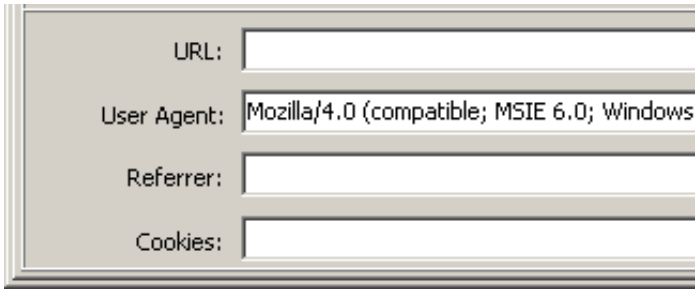
Mini HTML view – dummy render of the HTML document. Not very useful, except at rendering Apache's HTML presentation of FTP folders.

Middle button panel, part two:

Well, it is a classic – Find (Search) function.

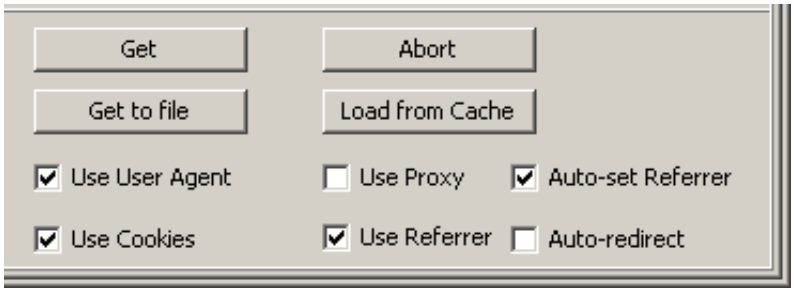
Format code will beautify the code (line breaks, indention) to make it easier to read.

Lower panel, part one:



The fields seen on this part of panel are to be set before requesting the document from the server as these are to be sent to the server in HTTP request.
User Agent list can be edited on Settings tab.

Lower panel, part two:



Get button will fetch the page/document for you.
Here are all the functions of the Get button:
HTTP URLs:
+ SHIFT = do not trim whitespaces from the end of URL

FTP URLs:
+ SHIFT = do not trim whitespaces from the end of URL
+ CTRL = LIST folder on FTP server (works only if URL points to a folder)
+ SHIFT + CTRL = no trim + LIST

Get to file – sometimes the web servers are not configured to send proper MIME types for binary files. This will cause Save File dialog not to appear if using normal **Get** button. **Get to file** will force the file to be saved, not just displayed on the screen.

Abort – abort the download

Load from cache – Malzilla will keep **every** fetched document in cache (until you clean the cache). You can use this button to load a file from the cache.

Options/checkboxes – pretty much self-explanatory, except of maybe the following two:
Auto-set Referrer – after fetching a document, the document's URL will be set as referrer for the next fetching.
Auto-redirect – Malzilla can detect redirections that are done through HTTP responses. Here you may decide if Malzilla should follow the redirections automatically, or to prompt. You may consider to set **Auto-set Referrer** too if you are using auto-redirections.
Malzilla does not detect redirections done through HTML/JS code. It is up to you to find such redirections in the code.

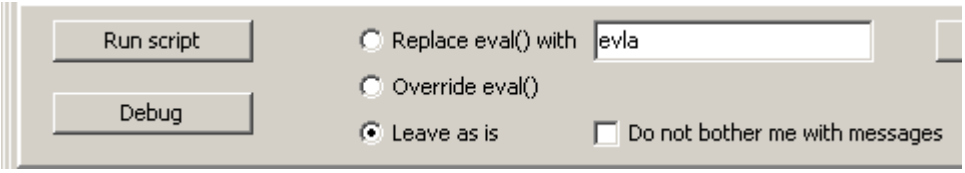
Decoder tab

First thing to clarify – you will not get infected by deobfuscating obfuscated scripts in Malzilla.

To deobfuscated a script we need to run the script. Every script around also contains self-deobfuscating routine. This will deobfuscated the rest of the script.
People are afraid to run the script because they think they will get infected that way. Well, no. Not running the script will infect the system, but what the script do with the objects from the browser.
Scripts are interacting with browser in both ways. A script can do such interaction with the browser, asking it to do something malicious.
Malzilla will do nothing, except printing on the screen all that the script is asking Malzilla to do.
The interaction model between a browser and the JavaScript is called DOM (Document Object Model): http://www.w3.org/DOM/
DOM is something that needs to be implemented on browser's side.
In Malzilla, there is just a basic DOM implemented. If the script is asking for DOM objects that are not implemented in Malzilla, you need to provide/emulate them by implementing them in JavaScript, in the same document where you have the script to deobfuscate.
Take a look at Templates in Decoder tab, you will see implementation of some DOM objects that are not provided by Malzilla. This way you can also emulate DOM implementation of various browsers (IE, Mozilla etc.)

So, let's take a look at the Decoder tab's interface.
In upper part you have tabbed interface (just like on Download tab).
Upper box is for entering the script.
Lower box is output box (results).

In the middle, there is a bunch of buttons, tiny boxes etc. Do not touch them, or a virus will eat all of your MP3 files ☺



Run script does exactly what it says – will run the script

Debug – if you get a error message at running the script, and you do not get any intermediate results from eval() monitoring, you can try this one.
It will open debugger window (read only, no interaction) if debugger caught the error at all. Please save all your data from Malzilla before using debugger – debugger is buggy and may freeze Malzilla (just killing the process from Task Manager helps).

Eval() options. Most annoying function that was a real headache for all the analysts. How to catch the results of eval() function?

There was a couple of methods to get the eval() results.
Most of the people were overriding eval() function by re-defining it like this:

```
function eval(a)
{
  document.write(a)
};
```

That's **Override eval()** in Malzilla, and it did worked in beginning until we got scripts that were checking for such overriding.

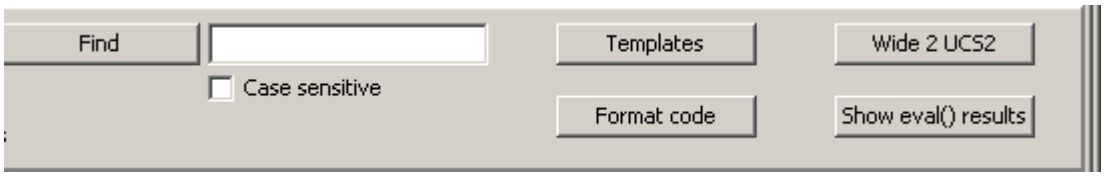
After that, there was my invention by dynamically adding new global method to JavaScript engine that would be used as callback for eval() functions.
That's **Replace eval() with** option in Malzilla.
That also worked some time until we encountered new generation of scripts.

Latest and the best method is to leave eval() in script as is, but hack JavaScript engine to use temporary files on HDD to store eval() results.
It takes time, it takes space on HDD, but it works (almost) every time.

So, use **Leave as is** as preferred option for getting eval() results.
If script produced eval() results while running, after script finishes a list of caught eval() outputs will be shown on the screen. Click on a result to see its content.

Do not bother me with messages is for **Override** and for **Replace** options.

Let's take a look at the right part of the panel:



We have there a classical Find function.

Templates – see the beginning of this chapter. You can have portions of code as templates. If you click on this button, a panel will appear on right side.
Click on item in panel to add a template code into the script.
The template code will be added at the beginning of the code.
Put your own templates into Templates folder to have them listed here.

Format code – pretty much the same like on Download tab.
You will probably want to use it also before running Debugger.

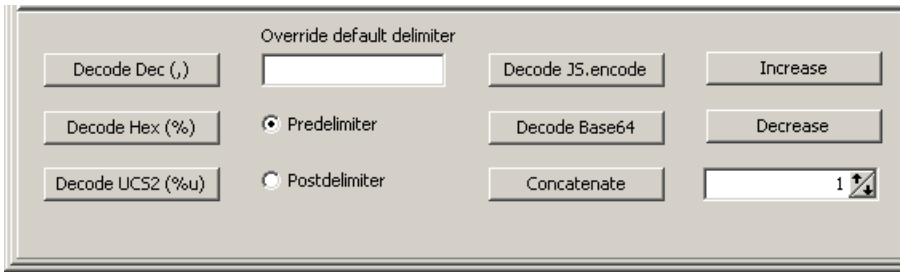
Wide 2 UCS2 – legacy from two years ago, when you needed to use this function on some scripts. If a script produces some garbage as output, you can try to fix it by using this button.

Show eval() results – will open the eval() results list if you closed it by mistake.

Misc Decoders tab

Misc Decoders tab contains two sub-tabs – Text and Hex.
The content of the two tabs is exchanged between tabs at changing the view from one to the second tab. That means, if you did any editing on Text tab, and you change the view to the Hex tab – Hex tab will fetch the content of the Text tab. If you edit something on Hex tab, and change the view to Text tab – Text tab will fetch the content of Hex tab.
At changing the view from Text to Hex, a routine will try to determine if the data on Text tab is Unicode or ASCII, so that it uses the right moving of data to Hex tab.
At changing from Hex to Text tab, the settings on lower part of the Hex tab are used for determining if the content is Unicode or not.
In Malzilla 1.1.0 and all the previous version, all the operations are done on data from the Text tab. This may change in the future, as Hex presentation of data is more reliable for doing operations.

Operations, part one:



Decode Dec will take decimal presentation of data, separated by a delimiter, and convert it to textual presentation.
Example: `65,65,65` will be converted to `AAA`

Decode Hex will do the same, but for hexadecimal presentation of data.
Example: `%41%41%41` will be converted to `AAA`

Decode UCS2 will take UCS2 input, and convert it to textual presentation.
Example: `%u03c4%u03b5` will be converted to `τε` (two Greek letters).
UCS2 is that that we usually (and wrong) refer to as Unicode.

HTML documents are using UCS2 encoding to present texts in languages other than Central European.
This is not necessary true if the HTML document is prepared on non-Windows platforms, as other platforms are using other Unicode encodings as default encodings.

Override default delimiter – as data to analyze may use different delimiters, by using this function you can use other than default delimiter for doing conversions. Default delimiters are shown on Decode buttons.
Example: `65#65#133` is decimal data, but it uses # as delimiter. Put # in Override default delimiter field, and click on Decode Dec after that.

Predelimiter/postdelimiter – decimal data are normally using a delimiter after the data member (e.g. `65,65,65`), but if you get data in e.g. following format `#65#65#65`, then you should use Predelimiter option to get the proper conversion. Hexadecimal and UCS2 are normally using predelimiter, but if you get Hex or UCS2 with delimiter after the data member, you can use this option for getting the proper conversion.

Robustnes of decoders – all 3 decoders are done so that no kind of data can throw an exception in conversion. It will ignore all the data that does not conform to the expected input.
Example: Doing decimal conversion on following data `65,t,67r,t65,62` will give the following output: `At,67r,t65,>`
Example 2: Hexadecimal conversion of `test%2Ethe_rest` will give `test.the_rest,` pretty much the same result you get from URL decoding functions in browsers.

Decode JS.encode – JScript.encode is IE-only function.
More info can be found here:
http://en.wikipedia.org/wiki/JScript.Encode
http://www.virtualconspiracy.com/content/articles/breaking-screnc
Decode JS.encode from Malzilla can do decoding only if original text (before encoding) was written just with ASCII characters.
It means, if the original text (before encoding) was written in (for example) Japanese, Malzilla can't decode it. In such cases please visit the second link mentioned above and get a stand alone version of decoder that can deal with codepages.

Decode Base64 – I will not explain this one. If you do not know what Base64 is, you are probably downloaded Malzilla by mistake, and you will newer be a reverser ☺

Concatenate – if you are serious user of Malzilla, you will probably use this function often than drinking a coffee.
Examples of concatenation conformant to what your browser would do:

```
"abc" + "de"      > "abcde"
'abc' + 'de'      > 'abcde'
"abc" & "de"      > "abcde"
'abc' & 'de'      > 'abcde'
```

Examples of non-conformant concatenation (but Malzilla will still concatenate):

```
"abc" + 'de'      > "abcde'
'abc' + "de"      > 'abcde"
"abc" & 'de'      > "abcde'
'abc' & "de"      > 'abcde"
```

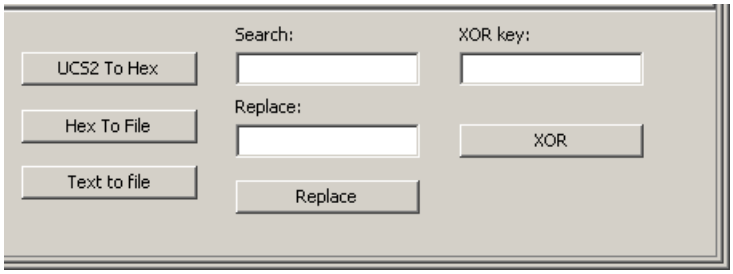
Resulting strings are not valid for browsers (mixing quotation and double quotation marks)

Example of a tricky concatenation that works in a browser:
`"abc"%2B"de"  > "abcde"`
In Malzilla, you will need to use Hex decoding, to get that `%2B` converted to +, and concatenate after that.

Increment/Decrement – increment or decrement the data, based on incrementing or decrementing according to ASCII values of data.
Used in some VBScript obfuscations in the past.

Important – the decoding functions, if accessed by clicking on buttons does not work on selections (selecting a part of data, and apply the function just to that part).
If you need to work with selections, please take a look at right-click menu explanation latter in this document.

Operations, part two:



UCS2 To Hex – you will need to use this if you want to run the shellcode from various exploits. After this conversion you can use the output result to run it in Shellcode Analyzer.

Hex To File – this is the next thing you will probably want to do after doing UCS2 To Hex. This will save the hex presentation from Misc Decoders to a binary file.
This function is deprecated after Hex tab is introduced in Misc Decoders.

Text To File – normal Save As function. Deprecated after implementing some options in right-click menu.

Search/Replace – do not need explanation.

XOR – doing binary XOR operation between the ASCII values of data and the XOR key. Deprecated by new functions in Hex view tab.

Kalimero Processor tab

I do not like when I need to include modules/functions that are limited to decoding of one specific exploit, but there was no other solution for dealing with LuckySploit.

Enter the whole HTML document into the left box.
Edit the regular expression under the box if default RegExpr does not match the HTML elements that we need to get.
Click on Detect.
If the operation was successful, in the right box you should see an array of element names and values.
Under the array box, there is another tiny box where you can define the form of output you need from array.
After that click on Make.
In the left box you will get the new array of functions that you can use in Decoder tab to deobfuscate LuckySploit. The array you get is to be used with LuckySploit template (look at Templates in Decoder).

Syntax for Regular Expressions used is this one:
http://RegExpStudio.com

You can use RegexBuddy to develop the expressions. RegExpStudio uses subset of Perl RegEx. Just a couple of Perl regular expressions forms do not apply to RegExpStudio’s set.

Shellcode analyzer tab

The Shellcode analyzer is based on libemu project:
http://libemu.mwcollect.org/

Load the shellcode into the upper box and click on Run emulation after that.
Libemu will run the shellcode in emulated environment (a virtual CPU) and it will hook the API calls done by the shellcode.
The set of API call hooks is limited to the most used ones, but that will do the job in 99% of the cases if you want to get the URL of the payload.

Shellcode analyzer implemented in Malzilla does not use the full power of libemu. You may want to take a look at sctest.exe in Malzilla/libemu folder if you need more advanced emulation.

GetPC – you may try this if the shellcode is not at the beginning of the file you loaded.

Log tab

On this tab you will find all the reports from logging function in Malzilla after you select **Log** option in right-click menu. More on logging later in this document.

Clipboard Monitor tab

If you do a right-click on Malzilla's tray icon, you can turn on and off the clipboard monitoring function.
Per default, this monitor will collect all the links from clipboard and add them to the list.
Useful if you get a list of links to analyze or to download.
As the clipboard monitor exposed a lot of bugs on some systems, you can add links to the list manually by using right-click menu on the list.
You can delete a link from the list by selecting it and pressing Del key on the keyboard.

By using the buttons on lower panel you can either just send the topmost link to the Downloader tab, send and Get the document from the topmost link, or use Download all to use Clipboard Monitor as a download manager (although not so perfect download manager as it does not follow all the redirections).

After a link is processed, it will be moved to lower box, including a report if the download was successful (server response code from HTTP headers).

Notes tab

Simple box for taking notes. Nothing extraordinary.

Hex view tab

Well, this is a beast of its own kind. It is meant primarily for working on shellcode, but can be used also to work on other kind of short binaries.

The screen is composed from 3 parts:

- Hex view with 3 view options
- XOR key finder, based on brute force
- Disassembler box

XOR key finder

If you get a piece of shellcode where a part of code is encrypted by a XOR operation, you may try to find the XOR key by doing a brute force string-search.
That means, XOR finder will try keys one after another (beginning with $00) until one of the strings we search for appears in the resulting code.
Example: you have a piece of shellcode, and you suspect that this shellcode downloads something from the net. So, you put the string HTTP in the **Strings to find** box, and click on **Find** after that.
You may optionally limit the search by setting **Max. Key to search.**
The search for strings is not case sensitive.

If a key is found, it will appear in **Key** box. If you click on **Apply XOR** the key will be used do a XOR operation on the data from the viewer.

Disassembler

Other way to find the XOR key is to disassemble the shellcode and hope that the XOR key can be easily spotted in the disassembler output.
The disassembler is based on libdisasm version 0.20:
http://bastard.sourceforge.net/libdisasm.html
Delphi port of libdisasm is done by Russell Libby.

PScript tab

I can't provide deobfuscation functions for all kind of obfuscations on the world.
PScript is a built-in Pascal interpreter. You can use it to write your own scripts.
A couple of example scripts are available in PScript folder in Malzilla.

Pscript will probably be removed from Malzilla in the future, or substituted by some more common scripting language interpreter (Perl, Python…)

Tools tab

EdEx

EdEx is a kind of text editor, designed with list of web links in mind.
It provides some functions to mass edit lists.

Numbered list maker

Another tool for managing download lists.
Did you ever got an info like "the folder is full with files named file1.exe to file35.exe"?
Well, if yes, then you know what this tool is mentioned for. It will make a download list for you.
Put one full link in upper box, enter the rest of the data in lower boxes (not all of them are needed for every case).

Example:

URL: `www.site.com/file001.exe`
Replace: `001`
With: `counter from 1 to 10`
Use padding to `3` (add zeros)

Will make the following list:

```
www.site.com/file001.exe
www.site.com/file002.exe
www.site.com/file003.exe
www.site.com/file004.exe
www.site.com/file005.exe
www.site.com/file006.exe
www.site.com/file007.exe
www.site.com/file008.exe
www.site.com/file009.exe
www.site.com/file010.exe
```

Templated list maker

This one was interesting some time ago…

Example:

Input box:
```
192.168.1.1
192.168.1.2
192.168.1.3
```

Template:
```
/file.exe
/ecard.exe
```

Will give the following output:
```
192.168.1.1/file.exe
192.168.1.1/ecard.exe
192.168.1.2/file.exe
192.168.1.2/ecard.exe
192.168.1.3/file.exe
192.168.1.3/ecard.exe
```

IP converter

Please read the following article:
http://www.searchlores.org/obscure.htm
(if the page goes offline, I have a local copy)

IP converter can convert all the forms mentioned in the article to usual URL form.

Settings tab

I'll not pay any attention to this tab as all the options should be self-explanatory if you read the rest of this document.
Only option that would need some explanation is **Add project info to the saved files**
This option will add additional info to the files saved from Download tab (if you use **Save to file** option from right-click menu).
Example of code added to the beginning of the file:

```
<!-- Malzilla Project v.1 -->
<!-- DAT: 25/10/2008 15:44:27 -->
<!-- URL: www.google.at -->
<!-- REF:  -->
<!-- UAS: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
.NET CLR 1.1.4322) -->
<!-- CCK:
PREF=ID=772c307c29f0a419:TM=1224942259:LM=1224942259:S=lTovG-
Mvj0gJhE7m -->
```

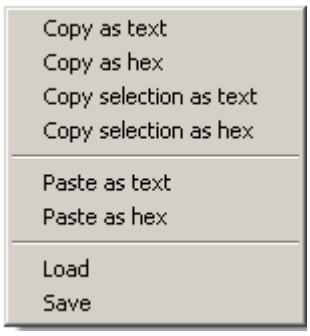
As it can be seen, all the project info is added in the form of HTML comments, so it will not do any harm to the original HTML code.
Next time you load this file in Malzilla, the info from the file will be parsed to the belonging boxes in Malzilla.

Right-click menu

Depending on where you do a right-click, there are a couple of menus that can appear:

- menu for hex view components
- menu for textual components
- tabs menu (already explained)

Menu for hex components:



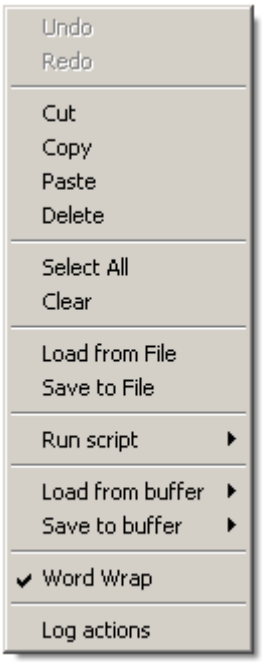
As text is always referring to the right part of hex view, and **as hex** is referring to the left part.
To understand the difference, please copy the following text:

```
4D616C7A696C6C61
```

Now, use **Paste as text** on a hex view component.
After that, use **Paste as hex** on the same hex view component.

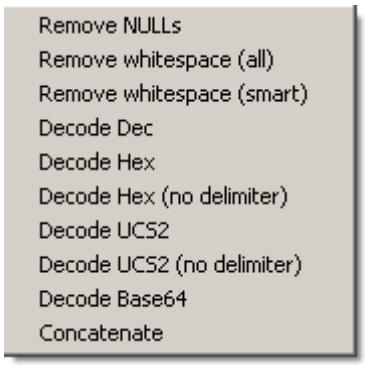
See the difference? Same goes for Copy functions.

Menu for textual components:



Undo is working just to undo the manual editing of the content.
Cut/Copy/Paste/Delete/Select All/Clear – standard text editing commands
Load from file/Save to file – as the names say
Run script – will be explained separately
Load from buffer/Save to buffer – obsolete as we have unlimited number of tabs in Download and Decoder now. There are 5 slots where you can temporarily save the content of the textual component. This will probably be removed from Malzilla in the future.
Word Wrap – usual option in every text editor.
Log actions – it will record your actions until you click it again. At finishing logging, it will ask you where to save the log. Log contains not just textual description of what you did, but also the files you got downloaded or decoded.

Run script sub-menu:



This sub-menu contains functions that are also to be found on Misc Decoders tab, and also some additional functions.
The scripts are also working on selections, e.g. if you have selected a part of document, the script will be run just on that part.
Input data is always overwritten by the output data.

Malzilla's cache

Malzilla's cache is nothing like some speed-up cache. It is more like an archive if you want to take a look at some documents that you downloaded earlier.
The cache is organized like following:

- folder Cache contains all the fetched files, all renamed to their MD5 hash sum
- files CacheMD5, CacheURL and CacheDate are in synchronization (e.g. line 10 from one file is related to line 10 from other file, and to line 10 from 3rd file). As the names of the files suggests, these are containing the MD5 of the fetched document, the Date and the URL. If these 3 files are out of sync (different number of list entries), a error message will appear.
- file CacheOther is not synchronized with other Cache* files. It contains various data indexed by a MD5 hash of a fetched document. It means, if you ask for a document from Cache list, Malzilla will load a file from cache according to the entry from CacheMD5. After that, it will search for same MD5 hash in CacheOther file. If the hash is found, from the same line it will load the following data: URL, date and time, User Agent string, cookies. The index (hash) is not exclusive, e.g. there can be more entries with the same hash (more than one site returned the same document). If that occurs, an error message will appear. This should be changed in the future (dialog that will let you chose which entry you want to load).

Templates

Templates are pieces of code that you can insert into the script you currently work on. Malzilla provide some variables to templates. The variables will be replaced with values of current parameters in Malzilla at the time of inserting the template into a script in decoder.

The list of variables:

```
malzilla.location.hash
malzilla.location.hostname
malzilla.location.href
malzilla.location.pathname
malzilla.location.port
malzilla.location.protocol
malzilla.location.search
malzilla.navigator.userAgent
malzilla.document.cookie
malzilla.document.domain
malzilla.document.referrer
malzilla.document.URL
```