

**Remote Control System**

Da Vinci

Integrating third party exploits

Ver 8.2.0

Exploit Repository:

The exploit repository of the RCS system is inside C:\RCS\DB\exploits directory on the backend machine. Every file inside that directory is an exploit. The name of each file is in the form ht-????-??? and must be unique. We usually put the year in the first four digits and a progressive number in the other three, but it can be anything you want.

Exploit Structure:

Each file inside the exploit repository is actually a zip archive that contains all the required scripts for the exploit creation and the description of the exploit itself. The structure of a generic exploit is as follow:

ht-0000-00

* info.yaml
* generation\_script

The info.yaml file is mandatory and represents the description of the exploit itself. The information inside this file are used by the console to display the description and the parameters needed to create the exploit.

All the other files inside the zip archive are used to generate the exploit when the user requests its creation.

The server will use the information found in the yaml file to execute the generation script with the proper parameters.

Exploit description:

The info.yaml file must follow this structure:

|  |  |
| --- | --- |
| version: | Version of the yaml file (must be 20120701) |
| id: | Is the unique id used by the system to identify this exploit. There should never be two exploit with the same id |
| name: | The name of the exploit used by the console in the combo box |
| category: | Available values are: :social, :private, :zeroday (the last one will be displayed in bold) |
| format: | It’s a yaml array containing the file extension that the exploit will create. (ie. If the explot is for Microsoft word the extension will be doc) |
| platform: | The platform on which the exploits run (e.g: windows, ios, osx, etc) |
| multiserver: | A Boolean value indicating if the exploit produces multiple files to be automatically put on the collector after the creation |
| embed: | A Boolean value indicating if the agent will be embedded in the output file or if will be put on the collector for download  |
| exec: | The command line executed by the system on exploit creation (see later) |
| params: | A yaml hash representing the parameters to be passed to the executed script. The key of the hash represent the type of the parameter and the value is the name displayed in the console. “string: URL” means that a string argument is needed and its called “URL” in the console“file: Document” means that the argument is a file and the console will ask to upload it on creation“combo:” is an array of values displayed in a combo box, useful if an exploit can behave in different ways based on fixed input values |
| description: | This is the extended description of the exploit used by the console when the exploit is selected. It is possible to use html codes. |

An example of an info.yaml file is as follows:

version: 20120701

id: HT-2012-009

name: Microsoft Word XP/2003/2007/2010 + Adobe Flash

category: :zeroday

format:

- doc

platform: windows

multiserver: true

exec: python exploit.py payload:http %URL% "%OUTPUT%" "%FILE%" "%FILENAME%" %AGENT% %OUTPUT\_SERVER%

params:

 file: Document

 string: URL

description: "<br><b>Output:</b> DOC file<br><br><b>Note:</b> The target must be connected to the internet when opening the document. <br><br><b>Platform:</b> Windows<br><br><b>Tested on:</b><br>Windows XP/Vista/7<br>Microsoft Office XP/2003/2007/2010 (full patched) - <b>Require Adobe Flash v11.1.102.55 or above for Internet Explorer</b>"

Exploit creation:

When the system starts the process to create an exploit, the zip archive is extracted inside a temporary directory and every operation happens inside that directory. The path of this directory is random, so the script must use relative paths to reference other files. At the end of the creation process the directory is deleted.

The system will take the “exec” string inside the yaml file and will perform some replacement on the variables it encounters. After the replacement the exec string is executed via shell. The supported variables are:

|  |  |
| --- | --- |
| %OUTPUT% | The path to the file that the script must produce in output (this file will be sent to the user of the console) |
| %OUTPUT\_SERVER% | The path to the file that the script must produce if there are some files to be put on the collector (if the exploits has a two-stage downloader) |
| %AGENT% | This is an input parameter and is the agent built by the system |
| %URL% | The url from which download the second stage (this is linked to the string parameter in the params section of the yaml) |
| %FILE% | Input file uploaded by the user (if requested by params) |
| %FILENAME% | The original filename uploaded by the user (if requested by params) |
| %COMBO% | The value selected by the user in the combobox (if requested by params) |

The generation script will be executed inside the temp directory and the output files will be taken by the system and sent to the user and to the collector based on the yaml description.

Usually the kind of exploits that have “download&execute” shellcode will produce an %OUTPUT\_SERVER% file that will be published via http on the collector and the shellcode will download the agent via the %URL% parameter.

The generation script can be written in java, python or ruby. It could also be written in any other languages that the backend machine understands. If there are any dependencies, you have to install it on the server or put them inside the zip archive and reference them with a relative path.