CERIAS Tech Report 2006-45

The Watermark Evaluation Testbed (WET)

by Oriol Guitart and Hyung Cook Kim and Edward J. Delp

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

The Watermark Evaluation Testbed (WET): New Functionalities

Oriol Guitart, Hyung Cook Kim and Edward J. Delp

Video and Image Processing Laboratory (VIPER)
School of Electrical and Computer Engineering
Purdue University
West Lafayette, Indiana USA

ABSTRACT

While Digital Watermarking has received much attention within the academic community and private sector in recent years, it is still a relatively young technology. As such, there are few accepted tools and metrics that can be used to validate the performance claims asserted by members of the research community and evaluate the suitability of a watermarking technique for specific applications. This lack of a universally adopted set of metrics and methods has motivated us to develop a web-based digital watermark evaluation system known as the *Watermark Evaluation Testbed* or *WET*. This system has undergone several improvements since its inception. The ultimate goal of this work has been to develop a platform, where any watermarking researcher can test not only the performance of known techniques, but also their own techniques. This goal has been reached by the latest version of the system. New tools and concepts have been designed to achieve the desired objectives. This paper describes the new features of *WET*. Moreover, we also summarize the development process of the entire project as well as introduce new directions for future work.

Keywords: digital watermarking, watermark evaluation, watermark benchmarking, image database, xml

1. INTRODUCTION

Digital watermarking is the practice of hiding a message in an image, audio, video or other digital media element. Since the late 1990s, there has been an explosion in the number of digital watermarking algorithms published [1–6]. The sudden increase is mostly due to the increase in concern over copyright protection of content [7]. Because new devices store content in digital form, there is no degradation in quality of data after a copy is made [8,9]. Currently, cryptographic techniques are the most popular method used to prevent piracy [10]. The problem with cryptographic techniques is that after the content is decrypted to be consumed, the content is no longer protected against piracy. Because digital watermarking can embed information related to the content in the digital media element and can be designed to survive many attacks, it is seen as a technique to complement cryptography in preventing piracy. Applications of watermarking [6,11–13] include broadcast monitoring, owner identification, proof of ownership, transaction tracking, authentication, copy control, and device control.

An important and often neglected issue in the design of digital watermarking methods is proper evaluation and benchmarking [14–16]. This lack of a proper evaluation in designing watermarking methods causes confusion among researchers and hinders the adoption of digital watermarking in various applications. With a well-defined benchmark, researchers and watermarking software developers would just need to provide a table of results, which would give a good and reliable summary of the proposed scheme performances for end users. To address this issue, several still image digital watermarking benchmarks have been proposed. These include StirMark [16–18], Certimark [19], Checkmark [20], Optimark [21, 22] and the *Watermark Evaluation Testbed (WET)* [23, 24]. In [23,24], different watermarking algorithms were also compared using the Taguchi loss function [25,26].

The main goal of *WET* is to develop a platform, where any watermarking researcher could (1) test the performance of known techniques, (2) test their own new techniques, and (3) obtain fair and reproducible

This work was supported by the Air Force Research Laboratory, Information Directorate, Rome Research Site, under research grant number F30602-02-2-0199. Address all correspondence to E. J. Delp, ace@ecn.purdue.edu.

comparisons of the results. We believe that the new version of *WET* has achieved this goal with the version described in this paper. XML was a important tool used to develop the new functionalities in *WET* and we believe it should be a basic tool for future development. XML concepts as used in *WET* will be discussed in this paper. This paper is organized as follows: In section 2, we summarize the different implementations and features of the previous versions of *WET* as well as the related architecture. Section 3 gives a description of the new features of the system. In section 4, we describe our ideas for this new version of *WET*, focusing on XML as main development tool. Finally, the conclusions are given in section 5.

2. ARCHITECTURE AND FEATURES SUMMARIZATION OF WET

In this section, we summarize the different features and tools that composed *WET* in [23, 24] as well as its architecture. For detailed descriptions of previous versions of *WET*, refer to [23, 24] .[*] The testbed consists of three major components: the Front End, the Algorithm Modules and the Image Database. Each component of *WET* will be described below. The previous version ran on a 2.4 GHz Pentium 4 computer using the Fedora Core 1.0 operating system[†].

2.1. Front End

The Front End is the end users' main interface into *WET*. The Front End provides a web interface whereby a user can select various tasks to be performed. The user interface abstracts much of the underlying architecture and allows the user to focus on the tasks to be preformed. The Front End consists of three major components: the web server, the database server, and the GIMP-Perl server. There are two front end versions supported by the previous version of *WET*, the *initial version* and *advanced version.* The *initial version* is intended for either first time users to familiarize themselves with the system or for users who want to get a feel for how watermarking works. The *advanced version* provides an extensive image database. All metrics reported in the *initial version* are also reported in the *advanced version.* The next two subsections give more details about these two options.

2.1.1. Initial Version

The *initial version* provides a static image database of about fifty images and a intuitive user interface. Figure 1(a) shows the watermarking steps for the *initial version.* Several watermarking algorithms and attack algorithms are available to the user. Several statistics are available including the CPU time used for embedding/detecting the watermark, the Mean Square Error (MSE) between the original and watermarked image, the difference image between the watermarked and the original image, and some specific statistics such as the correlation value, number of bit errors, threshold value as a part of the results.

2.1.2. Advanced Version

The *advanced version* provides an extensive image database. The user can select images with particular attributes, i.e. chrominance, resolution, height, width, and category to work with. The *advanced version* is available in two modes: interactive and batch mode.
The interactive mode allows the user to manually step through the process illustrated in Figure 1(b). The process of the interactive mode is similar to that of the initial version with some improvements:

1. Several images can be selected to insert a watermark
2. Multiple attack algorithms can be performed on the same watermarked images
3. For non-blind algorithms, the user can detect the presence of a watermark in all watermarked images against a single image or against as many as 25 images.

[*]A copy of these papers are located at http://www.datahiding.org/about.html.
[†]The system is located at http://www.datahiding.org. To obtain access to use *WET*, contact wetbug@ecn.purdue.edu.

The batch mode is the part of *WET* that allows users to make a more exhaustive evaluation of the chosen watermarking techniques. Unlike the interactive mode, batch mode does not run the processes at the time that the user submits the job. Instead, batch mode queues the user requests into a job and sends e-mail to the user when the job has completed. The e-mail contains a URL where the user can retrieve the evaluation results. Thereby, jobs can be lengthy and process much more data than an interactive display, where a user expects results to be available quickly. To submit a job using the batch mode, first the user chooses the images to watermark. Second, the user selects one or more watermarking techniques to be evaluated and specifies their watermark embedding parameters. Third, the user (optionally) chooses the attacks to be performed on the watermarked images. In the fourth step, the user selects the watermark detection parameters. Finally, the user submits the e-mail address to complete the procedure and submit the job.

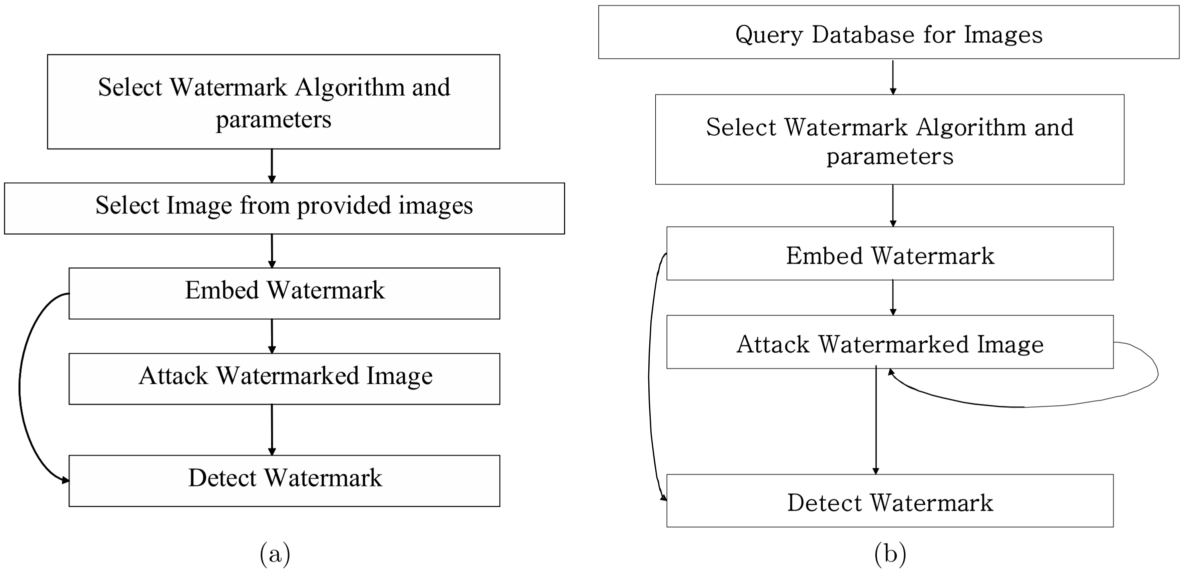

Select Watermark Algorithm and parameters
Select Image from provided images
Embed Watermark
Attack Watermarked Image
Detect Watermark
(a)
Query Database for Images
Select Watermark Algorithm and parameters
Embed Watermark
Attack Watermarked Image
Detect Watermark
(b)

Figure 1. Watermarking steps for: (a) Initial Version and (b) Advanced Version (Interactive Mode)

2.2. Algorithm Modules

It is desirable to develop tools that users can use standalone in their own test environments, allowing them to validate their tests locally before submitting them to a watermark benchmark site. To achieve this, the GNU Image Processing Program (GIMP) [27] was selected for use in *WET*. GIMP is designed to be augmented with plug-ins and extensions. Additionally, it provides full scripting support in various languages (Scheme, Perl, Java, Python, etc.). The combination of extensibility and scripting support make GIMP a powerful environment for *WET*.

Since the beginning of the project, we have been implementing several plug-ins for GIMP. Our plug-ins can be used in two different modes: Interactive Mode and Non-Interactive Mode. The former has a user interface where the user can choose the input parameters and the output parameters are displayed after using the plug-in. The Non-Interactive Mode performs the same function as the Interactive Mode but allows the plug-in to be called from another GIMP plug-in or scripts. The Non-Interactive Mode is used in *WET*. In the next subsections, we will summarize the plug-ins that have been implemented for *WET*, these plug-ins are divided between watermarking attacks and embed/detect watermarking techniques. For all the implemented plug-ins, we measure the computational complexity of the developed watermarking algorithm, this measure is obtained in terms of CPU execution time. Therefore, all GIMP plug-ins in *WET* were developed such that they return the CPU execution time as an output parameter.

2.2.1. Embed/Detect Watermarking Modules

In order to extend the implemented watermarking algorithms to color images, we use the reversible color transform (RCT) [28] developed for JPEG2000. The Red, Green and Blue components of an image are transformed by RCT and we embed the watermark into the luminance component. We chose RCT among different color transforms because it preserves the image when a watermark is not embedded. We watermarked the luminance component because a watermark should be placed in the perceptually most significant components of an image [3].

Since we started the *WET* project, we wanted to implement a wide variety of watermarking techniques and applications. We developed GIMP plug-ins for: non-blind [3, 29] and blind [3, 30–32] watermarking techniques, Semi-fragile watermarking techniques [33] and steganography [34–36] techniques. Finally, we also implemented a plug-in, which allows us to measure the fidelity between two images. We chose to measure the fidelity in terms of Mean Square Error (MSE). Mean Square Error is obtained as follows

$$MSE = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\left(p(i,j) - \hat{p}(i,j)\right)^2$$

where M is the number of color components, N is the number of pixels, p is the original image and $\hat{p}$ is the modified image.

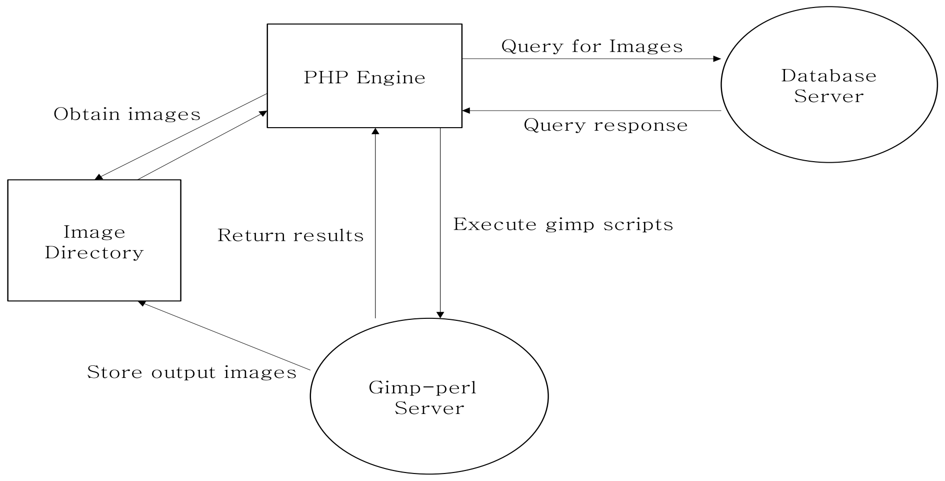


Figure 2. Advanced Version Architecture

2.2.2. Watermarking Attack Modules

We have implemented versions 3.1 and 4.0 of StirMark [17, 18, 37]. The default test mode of StirMark 3.1 was implemented as a GIMP plug-in. We implemented a GIMP plug-in for each defined attack in StirMark 4.0. The list of attacks in both versions is quite large and they could be classified into geometric transformations, signal processing operations and special transforms.

2.3. Image Database

The image database maintains the attributes of all images available in the test-bed. In the previous versions of *WET* we have approximately 1301 images which are copyright free. The images are from a variety of cameras and sensors including scanned photographs, x-ray images, ultrasound images, astrometrical images, line drawings, digital cameras, maps, and computer generated images. Each image in the database is stored with its attributes including chrominance, resolution, height, width, and category.

3. NEW FUNCTIONALITIES

We have integrated new functionalities into *WET* has allows any user to add new watermarking techniques to the system. The goal of the new developments is to provide a tool to allow anyone to demonstrate how their techniques perform against existing methods. At the same time, these new capabilities have a profound impact on the way *WET* is now developed, extended, and maintained.

The main new feature in *WET* allows users to design and integrate new watermarking modules into the system in a relatively easy way. In order to upload a new watermarking technique to the *WET* system, some additional information along with the implemented GIMP plug-in is required. Mainly, the required information needs to describe the new technique so that the system can upload the technique correctly for execution. This required information is provided by the user in what is known as the *module definition file* (MDF). A module is the collection of files for a particular watermark operation, these files include the GIMP plug-in, the MDF and other scripts. The MDF describes the structure of the GIMP plug-in as well as the significant elements of the user interface (such as description of all the parameters used by a plug-in, the help information and overview text, and special properties or restrictions for each parameter of the module.) The information stored in MDF will be used to create the necessary structure of files in the *WET* system so that the new technique can be incorporated into the system easily. There are many possible encodings for the MDF file. We have chosen XML [38,39] to implement the MDF file because of its simplicity and scalability properties. The next subsection will give complete details about the use of XML in this project as well as explain the development of the suitable XML schema. After this, we provide an explanation of the new architecture to fulfill these new functionalities. Finally, the last subsection describes the initial tests to verify the correct operation of the new tools.

3.1. XML Schema

The format and elements of the MDF file is a major design issue in the new developed tools. The MDF must be simple, concise, and extensible. It should also be a text file. After considering all these requirements, the XML language was chosen to encode the MDF file. The next step was to design the corresponding XML schema. An XML schema is a formalization of the constraints, expressed as rules or a model of structure that apply to XML documents. In many ways, XML schemas serve as design tools, establishing a framework on which implementations can be designed.

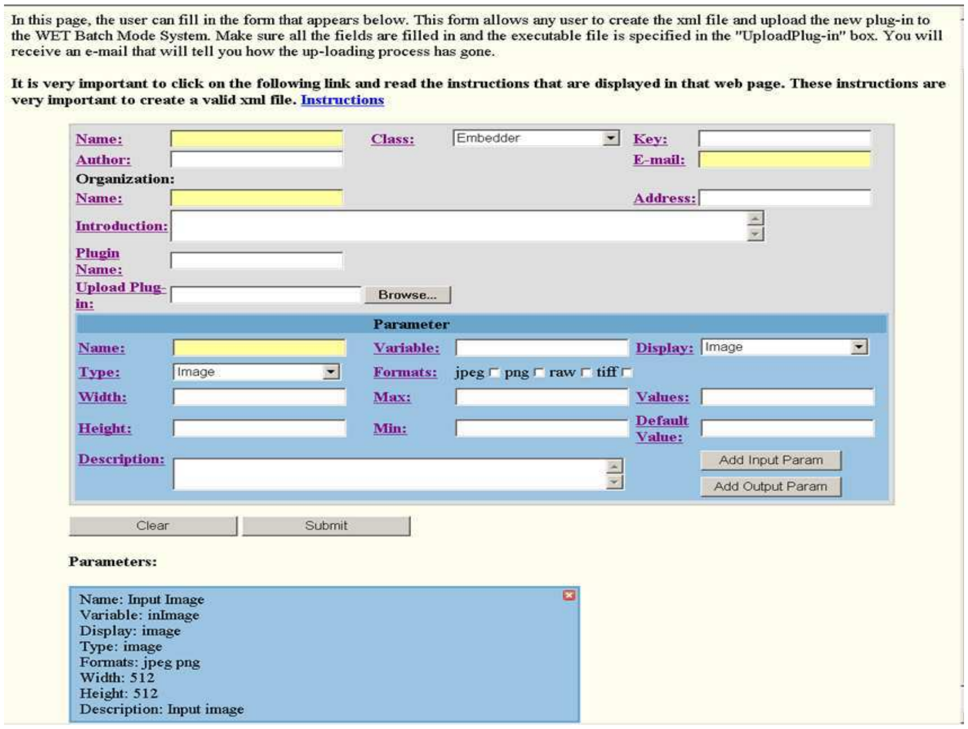
We have identified the following information requirements for the correct specification of the MDF:

- Determine the type of modules that the system must support.
- Determine common properties of the different types of modules.
- Determine the types of parameters that are needed for each type of module.
- Determine how other software may want to programmatically access the plug-in, generate user interfaces, etc.

After considering the requirements, several versions of the XML schema were developed. The final XML schema describes each possible type of plug-in that could be uploaded to *WET*. Even though our final XML schema is sufficient for the current requirements, the schema may undergo minor changes in the future to better support the needs of users. The following are the main features of the new schema:

- There are four possible modules: Embedding, Detection, Attacking and Evaluation.
- Each module has common information to be specified, such as the name of the plug-in, information describing the author(e.g. e-mail address) and description of the plug-in.
- All the parameters are considered to have common structure and are only differentiated between input and output parameters.

- Images are also considered as a type of parameter and have specific attributes such as width, height or format (e.g. jpeg, tiff). Some of this information may not be used in the current version but it is specified for future use.
- Each parameter needs to specify the user-interface control (such as edit box, list box) and the type (integer, float). This information will be used to request and manipulate correctly the parameter's value.
- For each parameter, the user may also specify the allowable range or set of values as well as the corresponding default value.


In this page, the user can fill in the form that appears below. This form allows any user to create the xml file and upload the new plug-in to the WET Batch Mode System. Make sure all the fields are filled in and the executable file is specified in the "UploadPlug-in" box. You will receive an e-mail that will tell you how the up-loading process has gone.

It is very important to click on the following link and read the instructions that are displayed in that web page. These instructions are very important to create a valid xml file. Instructions

Name:		Class:	Embedder	Key:	
Author:				E-mail:	
Organization:					
Name:				Address:	
Introduction:					
Plugin Name:					
Upload Plug-in:		Browse...			

Parameter

Name:		Variable:		Display:	Image
Type:	Image	Formats:	jpeg ☐ png ☐ raw ☐ tiff ☐		
Width:		Max:		Values:	
Height:		Min:		Default Value:	
Description:				Add Input Param	Add Output Param

Clear Submit

Parameters:

Name: Input Image
Variable: inImage
Display: image
Type: image
Formats: jpeg png
Width: 512
Height: 512
Description: Input image

Figure 3. WET Uploading Plug-in Interface

When the XML schema was designed, we considered a GIMP plug-in as a "black box" which accepts input, performs a useful task, and produces output. Thereby each element required for the plug-in is considered an input parameter in the MDF. At the same time, any element produced by the plug-in is considered an output parameter or result. For example, each MDF should have at least an image as an input parameter. On the other hand, an image do not always have to be specified as a result, since detection modules usually do not produce a new image. Figure 3 shows the web interface we developed to generate the MDF file. This is also useful since one can see the main features of the final XML schema.

3.2. Architecture

When we designed the architecture for the new functionalities, the main goal was to create an structure where the users should be able to access the new uploaded plug-in in exactly the same manner as they would access one of the currently existing plug-ins. After evaluating all the tools developed in the previous of *WET*, we chose the structure of batch mode to be the framework for the new uploaded plug-ins. Therefore, these new tools take advantage of the structure of batch mode and create the required elements so that the new uploaded plug-in can work as any other plug-in in the batch mode system.
Figure 2 shows the current structure of the batch mode system in the advanced version. This structure uses three types of files for each GIMP plug-in. A Perl script is used to communicate with the plug-ins and manage

the results. The PHP file has the interface information for each parameter. Finally, the HTML file has the code to display the help information for each parameter. After analyzing these files, we have identified a common structure for each file which allow us to create new files with the information contained in the MDF. We have developed a new tool known as the *module compiler* (MC) which will parse the MDF and produce the necessary scripts. Figure 4 shows the new structure with the MC inserted in the older structure of the advanced version.

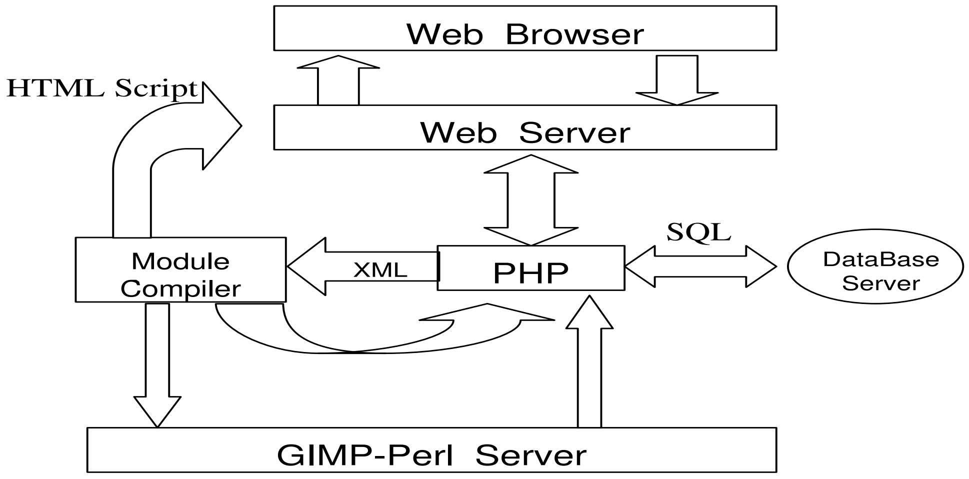


Figure 4. Uploading Plug-in Architecture

Once the new files have been successfully generated by the module compiler tool, they need to be uploaded to the system along with the executable GIMP plug-in. This process is known as the *module installer* and integrates the new plug-in into *WET*. The entire process is executed immediately so that if there are errors in some step of the process, the user will receive the corresponding feedback to fix the problem. We shall describe the sequences of steps that the process of uploading a plug-in must follow. If one step fails to perform the required action, the process starts again and the error is reported to the user.

1. **Create MDF**: The user must create the MDF with all the information of the new plug-in. Since creating the MDF may seem tedious, we have created a HTML web form inside the advanced version. This form allows the user to create the MDF file by entering the needed information in the form. Figure 3 shows a snapshot of the web form.

2. **MDF Verification**: There are two kinds of verifications that will be done to the MDF file. First, we have added constraints to the MDF generator (Web Form) which prevent the user from inserting incorrect information into the MDF. If some of the constraints are violated, the web form informs the user about the possible error. The second verification is done by the XML schema. We check the new MDF file (or XML file) against the XML schema. If the XML file is not valid according to the XML schema then the user is notified and the information must be corrected.

3. **Module Compiler**: Once the MDF is validated, the compiler parses the MDF and produces the required scripts (Perl, PHP, HTML). A tool was developed to fulfill this task. It examines an XML file and parses the information to generate the desired files. It uses several text templates and a XML library to generate the final scripts.

4. **Install new Plug-in**: When all the files have been generated, the new plug-in is ready to be uploaded to the system. First, the executable plug-in that must also be specified in the web form is stored in the corresponding plug-ins directory. The new files as well as the MDF file are stored in a new directory and its presence is flagged to the system. Finally, GIMP must be initialized so that the new plug-in is fully integrated to GIMP.

Figure 5 shows the process of uploading a plug-in. The step 4 of this process uses a queue to avoid possible conflicts when some of the installation tasks requires manipulating files that could be used for other tools of the system. The necessary initialization of GIMP in step 4 is also queued to allow the correct operation of the other tools. When the process of uploading a plug-in is successfully completed, the new MDF (XML file) is displayed.

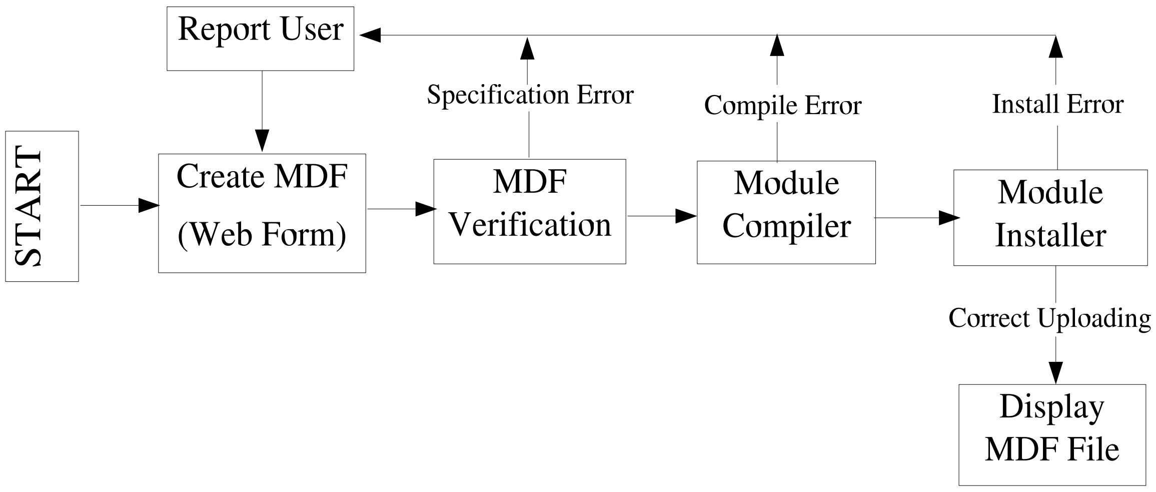


Figure 5. Uploading a Plug-in Steps

3.3. Experiments

In order to evaluate the reliability of the new tools, we tested them under different situations. First, we chose the spread spectrum watermarking technique [3] to check the new functionalities. This technique already has all the scripts (Perl, PHP, HTML) implemented and is fully operational in batch mode. Therefore, possible errors in the scripts were easily detectable and fixed. The rest of watermarking techniques in *WET*, such [32–34], which were in the previous version of *WET* but not available in batch mode, were uploaded to the batch mode structure. Finally, some of the implemented attacks in StirMark 4.0 [16–18, 37] were also uploaded to the batch mode system. We measured that the entire process of uploading a plug-in is on the order of a few seconds. Below shows a part of a XML file created to upload the embedder of the watermarking technique [3].

```
<?xml version="1.0" encoding="UTF-8"?>
<Module xmlns:xsi=" " xsi:noNamespaceSchemaLocation=" ">
        <Name>Embedding Cox Algorithm</Name>
        <Class>Embedder</Class>
        <Key>coxtest</Key>
        <Author>Ronaldinho Gaucho</Author>
        <E-mail>oguitart@detes.com</E-mail>
        <Organization>
                <Name>Viper Lab</Name>
                <Address>West Lafayette, IN USA</Address>
        </Organization>
        <Introduction>Thsi is the first test to use xml files to upload new modules.</Introduction>
        <PluginName>plug_in_sssw_embed_v2_2</PluginName>
        <Parameter>
                <Name>Input Image</Name>
                <Variable>inImage</Variable>
                <Display>image</Display>
                <Type>image</Type>
                <Description>This is the input image in the plug-in</Description>
        </Parameter>
        <Parameter>
                <Name>Scalar Factor</Name>
                <Variable>alpha</Variable>
                <Display>box</Display>
                <Type>float</Type>
                <Min>0</Min>
                <DefaultValue>0.1</DefaultValue>
                <Description> This parameter is the embedding strength.
                              The default value is 0.1.
                </Description>
        </Parameter>
        <Parameter>
                <Name>Key</Name>
                <Variable>key</Variable>
                <Display>box</Display>
                <Type>integer</Type>
                <Min>1</Min>
                <DefaultValue>123</DefaultValue>
                <Description>Normal random number generator
                             that has distribution N(0,1).
                </Description>
        </Parameter>
        <Parameter>
                <Name>Length</Name>
                <Variable>len</Variable>
                <Display>box</Display>
                <Type>integer</Type>
                <Min>1</Min>
                <DefaultValue>1000</DefaultValue>
                <Description>This is the number of DCT
                             coefficients that are watermarked.
                </Description>
        </Parameter>
</Module>
```

4. XML AND FUTURE WORK

The possibility of uploading new GIMP plug-ins will dramatically increase the amount of data needed to be processed and stored in the system. The storage and manipulation of this data is the main challenge of a future version of *WET*. The XML language has shown to be a very useful tool to deal with these types of problems. Working with XML has several benefits such as simplicity, openness and extensibility. After experimenting with XML, we have identified the following features which make this language very suitable for future versions of *WET*.

1. **Parsing Tools**: XML parsers present the data in a much more convenient format so that the user will have to write less custom code. There are a quite number of XML parsers written in languages such as Java, C, Perl and Phyton.
2. **Embedding Capacity**: It can embed multiple data types and cover all existing data structures.
3. **Efficient Storage**: XML databases store XML data natively in its structured, hierarchical form. Queries can be resolved much faster because there is no need to map the XML data tree structure to tables [40,41]. This preserves the hierarchy of the data and increases performance.

So far, several scripts are required for each module or plug-in defined, the amount of data could be reduced if we take advantage of the above defined properties. First, the large variety of XML parser tools will allow us to eliminate all intermediate scripts (Perl, PHP, HTML) and use directly the modules information from the XML files. Second, the image database should also be modified to fit into this new architecture. Therefore, we are considering ways to abstract the image database itself by using XML schema so that tools, software, and scripts can access images in a generic way [42]. Finally, instead of creating a large script with all the selected information for the batch mode jobs to be executed, a new XML file could be created to embed all the XML files related to the selected job, such as chosen images, selected attacks and so on. Then the engine file should only parse the information from this large XML file and generate the selected results. These modifications will require more work to design the additional XML schemas that better describe the possible applications. Moreover, some of the scripts already created will have to be modified to adapt them to the new structure.

5. CONCLUSION

In this paper, we described the new functionalities integrated into *WET*. The new features compared to the previous versions provide new motivation for watermarking evaluation.

The abstraction obtained by the XML schema is very efficient for extending the functionality of the system, while minimizing code development. However, new uploading capabilities will also bring new challenges for this system. Since the structure of *WET* will have to face with an increase in the number of modules or plug-ins it needs to manage. We believe that the properties of XML would be very useful to overcome these challenges and make the entire process more efficient.

REFERENCES

1. R. B. Wolfgang, C. I. Podilchuk, and E. J. Delp, "Perceptual watermarks for digital images and video," *Proceedings of the IEEE*, Vol. 87, No. 7, pp. 1108–1126, Jul. 1999.
2. E. T. Lin, A. M. Eskicioglu, R. L. Lagendijk, and E. J. Delp, "Advances in digital video content protection," *Proceedings of the IEEE*, Vol. 93, No. 1, pp. 171–183, January 2005.
3. I. Cox, J. Kilian, T. Leighton, and T. Shamoon, "Secure spread spectrum watermarking for multimedia," *IEEE Transactions on Image Processing*, Vol. 6, No. 12, pp. 1673–1687, 1997.
4. J. Fridrich and M. Goljan, "Comparing robustness of watermarking techniques," *Proceedings of the SPIE/IS&T Conference on Security and Watermarking of Multimedia Contents*, Vol. 3657, San Jose, CA, Jan. 1999, pp. 214–225.
5. C. I. Podilchuk and E. J. Delp, "Digital watermarking: Algorithms and applications," *IEEE Signal Processing Magazine*, Vol. 18, No. 4, pp. 33–46, Jul. 2001.
6. I. Cox, M. Miller, and J. Bloom, *Digital Watermarking.* Morgan Kaufmann, 2001.
7. E. J. Delp, "Is your document safe: An overview of document and print security," *NIP18: International Conference on Digital Printing Technologies*, San Diego, CA, Sep. 29–Oct. 4 2002, presented at.
8. A. M. Eskicioglu and E. J. Delp, "An overview of multimedia content protection in consumer electronics devices," *Signal Processing: Image Communication*, Vol. 16, pp. 681–699, 2001.
9. A. M. Eskicioglu, J. Town, and E. J. Delp, "Security of digital entertainment content from creation to consumption," *Signal Processing: Image Communication*, Vol. 18, pp. 237–262, 2003.
10. B. Schneier, *Applied Cryptography*, 2nd ed. New York: John Wiley and Sons, 1996.
11. I. J. Cox and M. L. Miller, "The first 50 years of electronic watermarking," *EURASIP Journal of Applied Signal Processing*, Vol. 2002, No. 2, pp. 126–132, 2002.
12. I. J. Cox, M. L. Miller, and J. A. Bloom, "Watermarking applications and their properties," *International Conference on Information Technology: Coding and Computing*, 2000, pp. 6–10.
13. F. Mintzer, G. Braudaway, and M. Yeung, "Effective and ineffective digital watermarks," *Proceedings of the IEEE International Conference on Image Processing*, Santa Barbara, CA, Oct. 1997, pp. 9–12.
14. B. Macq, J. Dittmann, and E. J. Delp, "Benchmarking of image watermarking algorithms for digital rights management," *Proceedings of the IEEE*, Vol. 92, No. 6, pp. 971–984, June 2004.
15. M. Kutter and F. A. P. Petitcolas, "A fair benchmark for image watermarking systems," *Proceedings of the SPIE/IS&T Conference on Security and Watermarking of Multimedia Contents*, P. W. Wong and E. J. Delp, Eds., Vol. 3657, San Jose, CA, Jan. 1999, pp. 226–239.
16. F. A. P. Petitcolas, M. Steinebach, F. Raynal, J. Dittman, C. Fontaine, and N. Fates, "A public automated web-based evaluation service for watermarking schemes: StirMark benchmark," *Proceedings of the SPIE/IS&T Conference on Security and Watermarking of Multimedia Contents*, P. W. Wong and E. J. Delp, Eds., Vol. 4314, San Jose, CA, Jan. 2001.
17. F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn, "Attacks on copyright marking systems," *Information Hiding, Second International Workshop*, D. Aucsmith, Ed. Portland, OR: Springer-Verlag, Apr. 1998, pp. 219–239.
18. F. A. P. Petitcolas, "Watermarking schemes evaluation," *IEEE Signal Processing Magazine*, Vol. 17, No. 5, pp. 58–64, Sep. 2000.
19. J. C. Vorbruggen and F. Cayre, "The Certimark benchmark: architecture and future perspectives," *IEEE International Conference on Multimedia and Expo*, Vol. 2, Lausanne, Switzerland, Aug. 2002, pp. 485–488.
20. S. Pereira, S. Voloshynovskiy, M. Madueño, S. Marchand-Maillet, and T. Pun, "Second generation benchmarking and application oriented evaluation," *Information Hiding Workshop*, Pittsburgh, PA, Apr. 2001.

21. V. Solachidis, A. Tefas, N. Nikolaidis, S. Tsekeridou, A. Nikolaidis, and P. Pitas, "A benchmarking protocol for watermarking methods," *Proceedings of the IEEE International Conference on Image Processing*, Vol. 3, Thessaloniki, Greece, Oct. 2001, pp. 1023–1026.
22. "http://poseidon.csd.auth.gr/optimark/."
23. H. C. Kim, H. Ogunleye, O. Guitart, and E. J. Delp, "The Watermark Evaluation Testbed (WET)," *Proceedings of the SPIE/IS&T Conference on Security and Watermarking of Multimedia Contents*, ser. Proceedings of SPIE Electronic Imaging, San Jose, CA, Jan. 2004, pp. 236–247.
24. H. C. Kim, E. T. Lin, O. Guitart, and E. J. Delp, "Further progress in Watermark Evaluation Testbed (WET)," *Proceedings of the SPIE/IS&T Conference on Security and Watermarking of Multimedia Contents*, ser. Proceedings of SPIE Electronic Imaging, San Jose, CA, Jan. 2005, pp. 236–247.
25. T. F. Rodriguez and D. A. Cushman, "Optimized selection of benchmark test parameters for image watermark algorithms based on Taguchi methods and corresponding influence on design decisions for real-world applications," *Proceedings of the SPIE/IS&T Conference on Security and Watermarking of Multimedia Contents*, P. W. Wong and E. J. Delp, Eds., Vol. 5020, San Jose, CA, Jan. 2003.
26. E. E. Lewis, *Introduction to Reliability Engineering.* John Wiley and Sons, Inc., 1996.
27. "http://www.gimp.org."
28. D. Taubman and M. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards, and Practice.* Kluwer Academic Publishers, 2002.
29. Y. Liu, B. Ni, X. Feng, and E. J. Delp, "Lot-based adaptive image watermarking," *Proceedings of the SPIE/IS&T Conference on Security and Watermarking of Multimedia Contents*, P. W. Wong and E. J. Delp, Eds., Vol. 5306, San Jose, CA, Jan. 2004.
30. A. Piva, M. Barni, F. Bartolini, and V. Cappellini, "DCT-based watermark recovering without restoring to the uncorrupted original image," *Proceedings of the IEEE International Conference on Image Processing*, Vol. 3, 1997, pp. 520–523.
31. H. S. Malvar and D. A. F. Florencio, "Improved spread spectrum: A new modulation technique for robust watermarking," *IEEE Transactions on Signal Processing*, Vol. 51, No. 4, pp. 898–905, Apr. 2003.
32. R. Dugad, K. Ratakonda, and N. Ahuja, "A new wavelet-based scheme for watermarking images," *Proceedings of the IEEE International Conference on Image Processing*, Chicago, IL, Oct. 1998.
33. E. T. Lin, C. I. Podilchuk, and E. J. Delp, "Detection of image alterations using semi-fragile watermarks," *Proceedings of the SPIE/IS&T Conference on Security and Watermarking of Multimedia Contents*, P. W. Wong and E. J. Delp, Eds., Vol. 3971, Jan. 2000, pp. 152–163.
34. B. Chen and G. W. Wornell, "Quantization index modulation methods for digital watermarking and information embedding of multimedia," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, Vol. 27, No. 1-2, pp. 7–33, Feb. 2001.
35. ——, "Quantization index modulation: A class of provably good methods for digital watermarking and information embedding," *IEEE Transactions on Information Theory*, Vol. 47, No. 4, pp. 1423–1443, May 2001.
36. E. T. Lin and E. J. Delp, "Locktography: Technical description," Internal Purdue memorandum.
37. "http://www.petitcolas.net/fabien/watermarking/stirmark/."
38. "http://www.w3.org/xml/."
39. E. R. Harold and W. S. Means, *XML in a Nutshell*, 3rd ed. O'Reilly Media, Inc., 2004.
40. T. Shimura, M. Yoshikawa, and S. Uemura, "Storage and retrieval of XML documents using object-relational databases," *Database and Expert Systems Applications*, 1999, pp. 206–217.
41. A. Vakali, B. Catania, and A. Maddalena, "XML data stores: Emerging practices," *Internet Computing, IEEE*, Vol. 9, No. 2, March-April 2005, pp. 62–69.
42. A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu, ""XML-QL: A Query Language for XML"," *WWW The Query Language Workshop (QL)*, Cambridge, MA, , 1998.
43. A. B. Watson, "DCT quantization matrices visually optimized for individual images," *SPIE Proceedings, Human Vision, Visual Processing and Digital Display IV*, J. P. Allebach and B. E. Rogowitz, Eds., Vol. 1913, San Jose, CA, Feb. 1993, pp. 202–216.