# Detection of Web Defacements by means of Genetic Programming

Eric Medvet Cyril Fillon Alberto Bartoli
*DEEI - University of Trieste, Via Valerio, 10, 34127 Trieste, Italy*
{emedvet, cfillon, bartolia}@units.it

## Abstract

*Web site defacement, the process of introducing unauthorized modifications to a web site, is a very common form of attack. Detecting such events automatically is very difficult because web pages are highly dynamic and their degree of dynamism may vary widely across different pages. In this paper we propose a novel detection approach based on Genetic Programming (GP), an established evolutionary computation paradigm for automatic generation of algorithms. What makes GP particularly attractive in this context is that it does not rely on any domain-specific knowledge, whose description and synthesis is invariably a hard job. In a preliminary learning phase, GP builds an algorithm based on a sequence of readings of the remote page to be monitored and on a sample set of attacks. Then, we monitor the remote page at regular intervals and apply that algorithm, which raises an alert when a suspect modification is found. We developed a prototype based on a broader web detection framework we proposed earlier and we tested our approach over a dataset of 15 dynamic web pages, observed for about a month, and a collection of real web defacements. We compared the results to those of a solution we developed earlier, whose design embedded a substantial amount of domain specific knowledge, and the results clearly show that GP may be an effective approach for this job.*

## 1. Introduction

Web site defacement is one of the most common form of attacks: it consists in replacing or modifying one or more web pages, either completely or only in part. More than 490,000 web sites have been *defaced* last year and the trend has been constantly growing in the recent years: every day, about 1500 web pages are defaced [11]. A defaced site may contain disturbing images or texts, political messages and so on, as well as a few messages or images representing a sort of signature of the hacker that performed the defacement. Fraudulent changes could also be aimed at remaining hidden to the user, focussing for example on links or forms.

The relative ease by which this sort of attack can be carried out makes the need for automated defacement detection techniques evident. What makes this job difficult is the very dynamic nature of web content: the challenge is how to deal with such highly dynamic content while keeping false positives to a minimum and, at the same time, while generating meaningful alerts.

In this paper we present a novel approach to automatic detection of web site defacement. This approach is based on *Genetic Programming*, an established evolutionary computation paradigm for automatic generation of algorithms which is briefly presented in Section 2. We use GP within a broader framework that we developed and refined earlier [2, 1]. We use a specialized tool for monitoring a collection of web pages, that are typically remote, hosted by different organizations and whose content, appearance, degree of dynamism are not known a priori. For each page, we execute a *learning phase* for constructing a profile that will then be used in the *monitoring phase*. When a reading does not fit the profile, the tool raises an alert—which could trigger the sending of a notification to the administrator of the page. The tool is modular in the sense that it delegates the details of learning and monitoring to pluggable modules. In the context of this paper, GP is the technology that we have used for designing and implementing such modules.

We evaluated the proposed approach on a dataset composed by 15 dynamic web pages that we observed for about a month. We simulated attacks by means of a set of real defacements extracted from a public attacks archive (Zone-H digital attack archive, http://www.zone-h.org). The effectiveness of GP on this task is remarkable: GP generates automatically algorithms capable of detecting almost all unauthorized modifications and coping with the highly dynamic nature of web pages while obtaining a false positive rate sufficiently low to be useful.

## 2. Genetic Programming in a nutshell

Genetic Programming (GP) is an automatic method for creating computer programs by means of artificial evolution [5]. A *population* of computer programs are gener-

0-7695-2876-7/07 $25.00 © 2007 IEEE
DOI 10.1109/IAS.2007.13

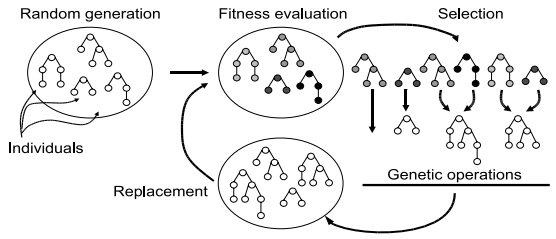


**Figure 1. The evolutionary cycle.**

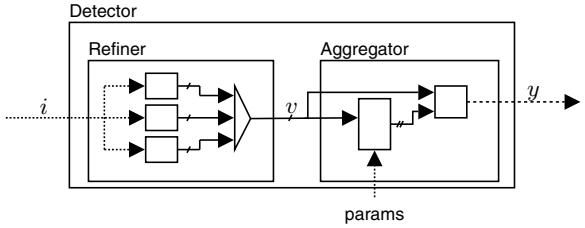


**Figure 2. Detector architecture. Different arrows types correspond to different type of data.**

ated at random starting from a user-specified set of building blocks. Each such program constitutes a candidate solution for the problem and is called an *individual*. The user is also required to define a *fitness function* for the problem to be solved. This function quantifies the performance of any given program—i.e., any candidate solution—for that problem (more details will follow). Programs of the initial population that exhibit highest fitness are selected for producing a new population of programs. The new population is obtained by recombining the selected programs through certain genetic operators, such as "crossover" and "mutation". This process is iterated for some number of generations, until either a solution with perfect fitness is found or some termination criterion is satisfied, e.g., a predefined maximum number of generations have evolved. The evolutionary cycle is illustrated in Fig. 1.

In many cases of practical interest individuals—i.e., programs—are simply formulas. Programs are usually represented as abstract syntax trees, where a branch node is an element from a *functions set* which may contain arithmetic, logic operators, elementary functions with at least one argument. A leaf node of the tree is instead an element from a *terminals set*, which usually contains variables, constants and functions with no arguments. Functions set and terminals set constitute the previously mentioned building blocks to be specified by the user. Clearly, these blocks should be expressive enough to represent satisfactory solutions for the problem domain. The population size is also specified by the user and depends on the perceived "difficulty" of the problem.

## 3. Related work

Several prior works have addressed the use of GP [10, 6], as well as other evolutionary computation techniques [9, 3], for network based or host based intrusion detection systems. What makes such approaches attractive is their ability to find automatically models capable of coping effectively with the huge amount of information to be handled [8]. We are not aware of any attempt of using such techniques for automatic detection of web defacements. One of the key differences between our scenario and such prior studies concerns the nature of the dataset used for training: in our case, it includes relatively few readings (some tens), each one composed by many values, whereas in the network and host-based IDS fields, it usually includes much more readings (thousands and more), each one composed by few values.

Concerning web defacements detection, there are several automatic tools tailored at solving this problem and some of them are commercially available. All the tools that we are aware of must be installed *on the site to be monitored* and are based on essentially the same idea: a copy of the page to be monitored is kept in a "very safe" location; the page content is compared to the *trusted copy* and an alert is generated whenever there is a mismatch [7, 4]. Such an approach has the potential to spot any unauthorized change, irrespective of how small and localized it is. On the other hand, it requires the site administrator to be able to provide a valid baseline for the comparison and keep it constantly updated. Yet, nowadays, most web resources are built on the fly dynamically, often by aggregating pieces of information from different sources, thus making this approach quite difficult to exploit in practice. We already proposed a significantly different approach to this problem, based on anomaly detection [2, 1]: for this paper analysis, we used the same pluggable framework we developed before for the cited works.

## 4. Scenario

We use GP within a tool that we developed earlier [1]. Full details about the tool can be found in the cited paper, we provide here only the context necessary for this work.

We consider a source of information producing a sequence of *readings* $\{i_1, i_2, \dots\}$ which is input to a *detector* (Fig. 2). The detector will classify each reading as being either normal (*negative*) or anomalous (*positive*). The detector consists internally of a *refiner* followed by an *aggregator*. In our scenario the source of information is a web page, univocally identified by an URL, and each reading consists of the document downloaded from that URL.

The refiner implements a function that takes a reading $i$ and produces a fixed size numeric vector $v = R(i)$. In our

case the transformation involves evaluating and quantifying many features of a web page related to both its content and appearance (e.g., number of external links, relative frequencies of HTML elements, and so on). The refiner is internally composed by one or more *sensors*. A sensor $S$ is a component which receives as input the reading $i$ and outputs a fixed size numeric vector $v_S$. The output of the refiner is composed by concatenating the output of all sensors. Our refiner produces a vector $v = R(i)$ of 1466 elements, obtained by concatenating the outputs from 43 different sensors. Sensors are functional blocks and have no internal state, that is, $v = R(i)$ depends only on the current input $i$ and does not depend on any prior reading.

The aggregator is the core component of the detector and it is the one that actually implements the GP approach. In a first phase, that we call the *learning phase*, the aggregator collects a sequence of readings in order to build the *learning sequence* $S_{\text{learning}}$. The GP process is applied on the learning sequence, as described below, with the purpose of obtaining an individual suitable for the detection task; during this phase, the aggregator is not able to classify readings. In a second phase, the *monitoring phase*, the aggregator uses the individual obtained earlier to analyze the current reading. In the monitoring phase, for each reading $i_k$ the aggregator may return either $y_k =$ negative (meaning the reading is normal) or $y_k =$ positive (meaning the reading is anomalous).

The GP process is actually applied at the end of the learning phase. Each individual implements a function $F(v)$ of the output $v$ of the refiner, for example ($v^i$ denotes the $i$-th component of $v$):

$$F(v) = 12 - \max(v^{57}, v^{233}) + 2.7v^{1104} - \frac{v^{766}}{v^{1378}} \quad (1)$$

The building blocks used for constructing individuals are described below. The output of the aggregator for a given reading $i_k$ is defined as follows ($v_k$ denotes the refiner output for the reading $i_k$):

$$y_k = \begin{cases} \text{negative} & \text{if } F(v_k) < 0 \\ \text{positive} & \text{if } F(v_k) \geq 0 \end{cases} \quad (2)$$

Individuals, i.e., formulas, of the population are selected basing on their ability to solve the detection problem, which is measured by the fitness function. In this case, we want to maximize the aggregator ability to detect attacks and, at the same time, minimize the number of wrong detections. For this purpose, we define a fitness function in terms of *false positive rate* (FPR) and *false negative rate* (FNR), as follows: (i) we build a sequence $S_{\text{learning}}$ of readings composed by readings of the observed page ($S_l$) and a sample set of attacks readings; (ii) we count the number of false positives—i.e., genuine readings considered as attacks—and false negatives—i.e., attacks not detected—raised by $F$ over $S_{\text{learning}}$, thus computing the respective FPR and FNR. Finally (iii), we set the fitness value $f(F)$ of the individual $F$ as follows:

$$f(F) = \text{FPR}(S_{\text{learning}}) + \text{FNR}(S_{\text{learning}}) \quad (3)$$

This fitness definition is applied by the GP process to select best individuals and repeat the evolution cycle, until either of the following holds: (1) a formula $F$ for which $f(F) = 0$ is found or (2) more than $n_{g,\max} = 100$ generations have evolved. In the latter case, the individual with the best (lowest) fitness value is selected.

The building blocks for individuals are: (i) a terminals set composed of $\mathcal{C} = \{0, 0.1, 1\}$, a specified set of constants, and $\mathcal{V}$, a specified set of independent variables from the output of the refiner; and (ii) a functions set composed of $\mathcal{F}$, a specified set of functions. We experimented with different sets $\mathcal{V}$ and $\mathcal{F}$ in order to gain insights into the actual applicability of GP to this task.

Concerning $\mathcal{F}$ we experimented with various subsets of $\mathcal{F}_{\text{all}} = \{+, -, \cdot, \div, \diagdown, \min, \max, \leq, \geq\}$, where $\diagdown$ represents the unary minus, and $\leq$ and $\geq$ returns 1 or 0 depending on whether the relation is true or not. All functions take two arguments, except for the unary minus.

Concerning $\mathcal{V}$ we experimented with various subsets of the set composed of all elements of $v$, the vector output by the refiner. To this end, we applied a *feature selection* algorithm for deciding which elements of $v$ should be included in $\mathcal{V}$. Note that elements in $v$ not included in $\mathcal{V}$ will have no influence whatsoever on the decision taken by the aggregator, because no individual will ever include such elements. The feature selection algorithm is aimed at selecting those $v$ elements which seem to indeed have significance in the decision and works as follows.

Let $S_{\text{learning}} = \{v_1, \ldots, v_n\}$ be the learning sequence, including all genuine readings and all simulated attacks. Let $X_i$ denote the random variable whose values are the values of the $i$-th element of $v$ (i.e., $v^i$) across all readings of $S_{\text{learning}}$. Recall that there are 1466 such variables because this is the size of $v$. Let $Y$ be the random variable describing the desired values for the aggregator: $Y = 0, \forall$ genuine reading $\in S_{\text{learning}}$; $Y = 1$ otherwise, i.e., $\forall$ simulated attack reading $\in S_{\text{learning}}$. We computed the absolute correlation $c_i$ of each $X_i$ with $Y$ and, for each pair $\langle X_i, X_j \rangle$, the absolute correlation $c_{i,j}$ between $X_i$ and $X_j$. Finally, we executed the following iterative procedure, starting from a set of unselected indexes $I_U = \{1, \ldots, 1466\}$ and an empty set of selected indexes $I_S = \emptyset$: (1) we selected the element $i \in I_U$ with the greatest $c_i$ and moved it from $I_U$ to $I_S$; (2) $\forall j \in I_U$, we set $c_j := c_j - c_{i,j}$. We repeated the two steps until a predefined size $s$ for $I_S$ is reached. Set $\mathcal{V}$ will include only those elements of vector $v$ whose indexes are in $I_S$. In other words, we take into account only those indexes with maximal correlation with the desired output (step 1),

attempting to filter out any redundant information (step 2).

## 5. Experiments and results

### 5.1. Dataset

In order to perform our experiments, we built a dataset as follows. We observed 15 web pages for about one month, collecting a reading for each page every 6 hours, thus totalling 125 readings for each web page. These readings compose the *negatives sequences*—one negative sequence $S_{N,p}$ for each page $p$: we visually inspected them in order to confirm the assumption that they are all genuine. The observed pages differ in size, content and dynamism and include pages of e-commerce web sites, newspapers web sites, and alike. They are the same pages that we observed in [1]. Then we built a single *positives sequence* $S_P$ composed by 75 readings extracted from a publicly available defacements archive (`http://www.zone-h.org`).

### 5.2. Methodology

In order to set a baseline for assessing the performance of GP, we injected the very same dataset to another aggregator that we developed earlier, not based on GP [1]. This aggregator implements a form of anomaly detection based on domain-specific knowledge. In short, it builds a profile of the observed page from a set of normal observations; then, it signals an anomaly whenever the actual observation of the page deviates from the profile, on the assumption that any anomaly represents evidence of an attack. Space constraints preclude a detailed description of this aggregator, we can only point out that its notion of "profile" encompasses many different points of view. For example, mean and standard deviation of the number of lines; set of images or links contained in every reading; subtree of the HTML tree contained in every reading. This aggregator raises an alert depending on number and types of deviations from the profile (see the cited paper for more details). Note that this aggregator makes use of a learning sequence that does not include any attack, thus it does not exploit any information related to positive readings. The GP-based aggregator, in contrast, does use such information. Note also that our existing aggregator takes into account all the 1466 elements output by the refiner, whereas GP-based aggregator considers only those elements chosen by the feature selection algorithm.

We generated 25 different GP-based aggregators, by varying the number of selected features $s$ in 10, 20, 50, 100, 1466 (thus including the case in which we did not discard any feature) and the functions set $\mathcal{F}$ in $\mathcal{F}_1 = \{+, -\}$, $\mathcal{F}_2 = \{+, -, \cdot, \div, \setminus\}$, $\mathcal{F}_3 = \{+, -, \cdot, \div, \setminus, \leq, \geq\}$, $\mathcal{F}_4 = \{+, -, \cdot, \div, \setminus, \min, \max\}$, $\mathcal{F}_5 = \{+, -, \cdot, \div, \setminus, \min, \max, \leq, \geq\}$.

We used FPR and FNR as performance indexes, that we evaluated as follows. First, we built a sequence $S'_P$ of positive readings composed by the first 20 readings of $S_P$. Then, for each page $p$, we built the learning sequence $S_{\text{learning}}$ and a testing sequence $S_{\text{testing}}$ as follows. (1) We split $S_{N,p}$ in two portions $S_l$ and $S_t$, composed by 50 and 75 readings respectively. (2) We built the learning sequence $S_{\text{learning}}$ appending $S'_P$ to $S_l$. (3) We built the testing sequence $S_{\text{testing}}$ appending $S_P$ to $S_t$. Finally, we tuned the aggregator being tested on $S_{\text{learning}}$ and we tested it on $S_{\text{testing}}$. To this end, we counted the number of false negatives—i.e., missed detections—and the number of false positives—i.e., legitimate readings being flagged as attacks. As already pointed out, the anomaly-based aggregator executes the learning phase using only on $S_l$ and ignoring $S'_P$.

In the next sections we present FPR and FNR for each aggregator, averaged across the 15 pages of our dataset. GP-based aggregators will be denoted as GP-$s$-$\mathcal{F}_i$, where $s$ is the number of selected features and $\mathcal{F}_i$ is the specific set of functions being used; anomaly-based aggregator will be denoted as Anomaly.

### 5.3. Results

Table 1 summarizes our results. The table shows FPR, FNR and the fitness exhibited by the individual selected to implement the GP-based aggregator (the meaning of the three other columns is discussed below). The aggregator with best performance, in terms of FPR + FNR, is highlighted. It can be seen that the GP process is quite robust with respect to variations in $s$ and $\mathcal{F}$. Almost all GP-based aggregators exhibit a FPR lower than 0.36% and a FNR lower than 1.87%. The anomaly-based aggregator—i.e., the comparison baseline—exhibits a slightly higher FPR (1.42%) and a slightly lower FNR (0.09%). In general, the genetic programming approach seems to be quite effective for detecting web defacements.

We analyzed GP-based aggregators also by looking at the number of generations $n_g$ that have evolved for finding the best individual and the complexity of the corresponding abstract syntax tree, in terms of number of nodes $t_s$ and height $t_h$ (these data are shown in Table 1). We found that formulas tended to be quite simple, i.e., the corresponding trees exhibited low $t_s$ and $t_h$. We also found, to our surprise, that $n_g = 1$ in most cases. This means that generating some random formulas (500 in our experiments) from the available functions and terminals sets suffices to find a formula with perfect fitness—i.e., one that exhibits 0 false positives and 0 false negatives over the learning sequence. We believe this depends on the high correlation between some elements of the vector output by the refiner (i.e., some $v^i$) and the de-

**Table 1. Performance indexes. FPR, FNR and $f$ are expressed in percentage.**

| Aggregator | FPR | FNR | $f$ | $n_g$ | $t_s$ | $t_h$ |
|---|---|---|---|---|---|---|
| Anomaly | 1.42 | 0.09 | - | - | - | - |
| GP-10-$\mathcal{F}_1$ | 0.00 | 0.71 | 0.0 | 1.0 | 17.0 | 2.7 |
| GP-10-$\mathcal{F}_2$ | 0.09 | 0.98 | 0.0 | 1.0 | 23.3 | 3.7 |
| GP-10-$\mathcal{F}_3$ | 0.09 | 0.62 | 0.0 | 1.1 | 20.4 | 3.5 |
| GP-10-$\mathcal{F}_4$ | 4.53 | 0.44 | 0.0 | 1.0 | 20.7 | 3.7 |
| GP-10-$\mathcal{F}_5$ | 0.09 | 0.89 | 0.0 | 1.0 | 27.9 | 3.9 |
| GP-20-$\mathcal{F}_1$ | 0.09 | 1.16 | 0.0 | 1.0 | 18.2 | 2.6 |
| GP-20-$\mathcal{F}_2$ | 0.18 | 1.33 | 0.0 | 1.0 | 12.8 | 2.4 |
| GP-20-$\mathcal{F}_3$ | 0.36 | 0.80 | 0.0 | 1.0 | 20.1 | 3.1 |
| GP-20-$\mathcal{F}_4$ | 0.09 | 0.89 | 0.0 | 1.0 | 36.5 | 4.2 |
| GP-20-$\mathcal{F}_5$ | 0.00 | 0.89 | 0.0 | 1.0 | 39.5 | 3.9 |
| GP-50-$\mathcal{F}_1$ | 0.00 | 1.24 | 0.0 | 1.0 | 5.1 | 1.6 |
| GP-50-$\mathcal{F}_2$ | 0.09 | 0.98 | 0.0 | 1.0 | 20.4 | 2.9 |
| GP-50-$\mathcal{F}_3$ | 0.36 | 0.98 | 0.0 | 1.0 | 19.3 | 2.9 |
| GP-50-$\mathcal{F}_4$ | 0.18 | 0.89 | 0.0 | 1.0 | 15.4 | 3.1 |
| **GP-50-$\mathcal{F}_5$** | **0.18** | **0.27** | **0.0** | **1.0** | **29.4** | **3.0** |
| GP-100-$\mathcal{F}_1$ | 0.09 | 1.16 | 0.0 | 1.0 | 15.5 | 2.1 |
| GP-100-$\mathcal{F}_2$ | 0.09 | 1.33 | 0.0 | 1.1 | 11.4 | 2.2 |
| GP-100-$\mathcal{F}_3$ | 0.00 | 1.87 | 0.0 | 1.3 | 14.1 | 3.1 |
| GP-100-$\mathcal{F}_4$ | 0.09 | 0.27 | 0.0 | 1.1 | 18.7 | 3.1 |
| GP-100-$\mathcal{F}_5$ | 0.09 | 1.33 | 0.0 | 1.2 | 15.4 | 2.6 |
| GP-1466-$\mathcal{F}_1$ | 0.00 | 0.80 | 0.0 | 1.0 | 8.9 | 1.9 |
| GP-1466-$\mathcal{F}_2$ | 0.18 | 0.44 | 0.0 | 1.0 | 6.1 | 2.3 |
| GP-1466-$\mathcal{F}_3$ | 0.18 | 0.98 | 0.0 | 1.2 | 5.3 | 1.5 |
| GP-1466-$\mathcal{F}_4$ | 0.18 | 1.87 | 0.0 | 1.2 | 11.2 | 1.8 |
| GP-1466-$\mathcal{F}_5$ | 3.73 | 0.18 | 0.0 | 1.4 | 9.9 | 2.1 |

sired output. Since the feature selection chooses elements based on their correlation with the desired output, most of the variables available to GP will likely have an high correlation with output.

Our domain knowledge, however, suggests that the simple formulas found by the GP process could not be very effective in a real scenario. Since they take into account very few variables, an attack focussing on the many variables ignored by the corresponding GP aggregators would go undetected. This consideration convinced us to develop a more demanding testbed, as follows.

### 5.4. Results with "shuffled" dataset

In this additional set of experiments, we augmented the set of positive readings for any given page $p_i$ by including genuine readings of *other* pages. While the previous experiments evaluated the ability to detect manifest attacks (defacements extracted from Zone-H), here we also test the ability to detect innocent-looking pages that are different from the usual appearance of $p_i$. More in detail, for a given page $p_i$ we defined a sequence $S^o_{\text{learning}}$ composed by 14 genuine readings of the *other* pages (one for each other page) and a sequence $S^o_{\text{testing}}$ composed by 70 readings of the other pages (5 readings for each other page, such that $S^o_{\text{learning}}$ and $S^o_{\text{testing}}$ have no common readings). Then, we included $S^o_{\text{learning}}$ in $S_{\text{learning}}$ and $S^o_{\text{testing}}$ in $S_{\text{testing}}$ (we omit the obvious details for brevity). Clearly, readings in $S^o_{\text{learning}}$ were

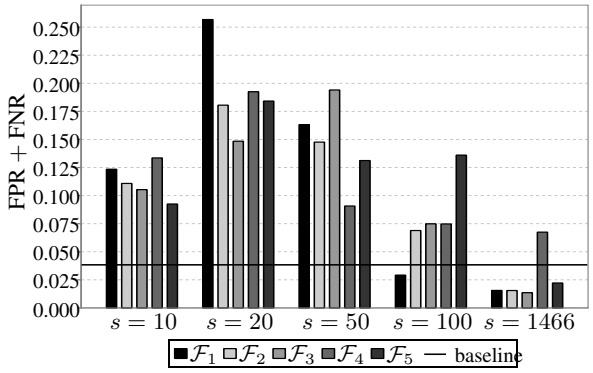


**Figure 3. Sum of FPR and FNR for different parameter combinations.**

labelled as positives and readings in $S^o_{\text{testing}}$ should raise an alarm.

Table 2 presents the results for this testbed. The anomaly-based aggregator now exhibits a slightly higher FNR; FPR remains unchanged, which is not surprising because this aggregator uses only the negative readings of the learning sequence and these are the same as before. We note that the anomaly-based aggregator is very effective also in this new setting, in that it still exhibits a very low FPR while being capable of flagging as positive most of the genuine readings of *other* pages.

Concerning GP-based aggregators, Table 2 suggests several important considerations. In general the approach seems now to be influenced by the number $s$ of variables selected: taking more variables into account lead to better performance, in terms of FPR + FNR. Interestingly, the best result is obtained with $s = 1466$, i.e., with *all* variables available to the GP process. Moreover, there are situations in which the fitness $f$ of the selected formula is no longer perfect (i.e., equal to 0). This means that, in these situations, the GP process is no longer able to to find a formula that exhibits $f = \text{FPR} + \text{FNR} = 0$ on the learning sequence. Note that this phenomenon tends to occur with low values of $s$. Finally, values of $t_s$, $t_h$ and, especially, $n_g$, are greater than in the previous testbed, which confirms that GP process is indeed engaged. Several generations are necessary to find a satisfactory formula and the formula is more complex than those previously found, in terms of size and height of the corresponding abstract tree.

Figure 3 shows results of Table 2 in a graphical way and visually confirms that the GP approach is quite robust in respect to the specific set of functions being used but is sensible to the number $s$ of selected features.

Finally, we compared the computation times for learning and monitoring phases obtained with GP-based approach against those of anomaly-based approach. The former takes about 100secs for performing the tuning procedure (of which about 5secs are used for the features selec-

**Table 2. Performance indexes with the new testbed. FPR, FNR and $f$ are expressed in percentage.**

| Aggregator | FPR | FNR | $f$ | $n_g$ | $t_s$ | $t_h$ |
|---|---|---|---|---|---|---|
| Anomaly | 1.42 | 2.39 | - | - | - | - |
| GP-10-$\mathcal{F}_1$ | 9.87 | 2.48 | 0.4 | 35.5 | 83.8 | 7.2 |
| GP-10-$\mathcal{F}_2$ | 9.24 | 1.84 | 0.0 | 14.3 | 69.1 | 7.5 |
| GP-10-$\mathcal{F}_3$ | 9.24 | 1.29 | 0.1 | 35.7 | 66.4 | 8.0 |
| GP-10-$\mathcal{F}_4$ | 11.38 | 1.98 | 0.1 | 24.1 | 93.1 | 7.9 |
| GP-10-$\mathcal{F}_5$ | 7.73 | 1.52 | 0.1 | 30.5 | 66.1 | 6.9 |
| GP-20-$\mathcal{F}_1$ | 23.38 | 2.30 | 0.1 | 16.9 | 54.2 | 5.3 |
| GP-20-$\mathcal{F}_2$ | 16.44 | 1.61 | 0.0 | 10.8 | 68.3 | 6.2 |
| GP-20-$\mathcal{F}_3$ | 13.51 | 1.33 | 0.0 | 16.4 | 52.1 | 6.2 |
| GP-20-$\mathcal{F}_4$ | 17.87 | 1.38 | 0.0 | 14.6 | 56.3 | 6.3 |
| GP-20-$\mathcal{F}_5$ | 17.87 | 0.55 | 0.0 | 19.5 | 70.4 | 6.5 |
| GP-50-$\mathcal{F}_1$ | 14.76 | 1.56 | 0.1 | 23.7 | 43.1 | 4.4 |
| GP-50-$\mathcal{F}_2$ | 13.16 | 1.61 | 0.0 | 4.7 | 45.0 | 5.4 |
| GP-50-$\mathcal{F}_3$ | 18.58 | 0.83 | 0.0 | 11.7 | 32.4 | 5.4 |
| GP-50-$\mathcal{F}_4$ | 7.56 | 1.52 | 0.0 | 12.5 | 41.1 | 6.0 |
| GP-50-$\mathcal{F}_5$ | 11.47 | 1.66 | 0.0 | 16.1 | 71.2 | 6.8 |
| GP-100-$\mathcal{F}_1$ | 0.62 | 2.30 | 0.0 | 29.4 | 51.5 | 4.5 |
| GP-100-$\mathcal{F}_2$ | 5.51 | 1.38 | 0.0 | 10.5 | 25.6 | 4.1 |
| GP-100-$\mathcal{F}_3$ | 6.93 | 0.55 | 0.0 | 16.4 | 33.9 | 4.8 |
| GP-100-$\mathcal{F}_4$ | 5.87 | 1.61 | 0.0 | 10.2 | 40.3 | 5.1 |
| GP-100-$\mathcal{F}_5$ | 12.09 | 1.52 | 0.0 | 17.7 | 31.9 | 5.2 |
| GP-1466-$\mathcal{F}_1$ | 0.18 | 1.38 | 0.0 | 21.0 | 37.4 | 3.8 |
| GP-1466-$\mathcal{F}_2$ | 0.44 | 1.10 | 0.0 | 18.5 | 24.8 | 4.1 |
| **GP-1466-$\mathcal{F}_3$** | **0.71** | **0.64** | **0.0** | **15.1** | **30.1** | **4.7** |
| GP-1466-$\mathcal{F}_4$ | 5.69 | 1.06 | 0.0 | 19.7 | 27.9 | 4.7 |
| GP-1466-$\mathcal{F}_5$ | 0.98 | 1.24 | 0.0 | 16.5 | 69.9 | 5.5 |

tion) and about $500\mu$secs for evaluating one single reading in the monitoring phase; the latter takes about 10msecs for the tuning procedure and about $100\mu$secs for evaluating one single reading in the monitoring phase. These numbers are obtained with a dual AMD Opteron 64 with 8GB RAM running a Sun JVM 1.5 on a Linux OS.

## 6. Concluding remarks

We have proposed and evaluated experimentally an approach based on Genetic Programming for detecting automatically defacements of web pages. What makes this problem difficult is that web pages are highly dynamic and the degree of dynamism change widely across pages. The main power of GP lies in its ability to construct automatically algorithms capable of describing the dynamic nature of a given web page without any domain-specific knowledge. We tested a prototype over a selection of 15 highly dynamic web pages that we observed for about a month and found that this approach is indeed practical: it is able to detect nearly all of the attacks that we simulated, while keeping the number of false positives sufficiently low to be practical. The approach exhibits performance close or better than other approaches that we pursued in the past, whose design required a considerable amount of domain-specific knowledge.

## Acknowledgments

This work was supported by the Marie-Curie RTD network AI4IA, EU contract MEST-CT-2004-514510 (December 14th 2004).

## References

[1] A. Bartoli and E. Medvet. Anomaly-based Detection of Web Site Defacements. *In submission*, 2006. Available at `http://www.units.it/~bartolia/abstract/AnomalyBasedDetectionOfWebSiteDefacements.pdf`.

[2] A. Bartoli and E. Medvet. Automatic Integrity Checks for Remote Web Resources. *IEEE Internet Computing*, 10(6):56–62, 2006.

[3] Y. Chen, A. Abraham, and B. Yang. Hybrid flexible neural-tree-based intrusion detection systems. *International Journal of Intelligent Systems*, 22(4):337–352, 2007.

[4] W. Fone and P. Gregory. Web page defacement countermeasures. In *Proceedings of the 3rd International Symposium on Communication Systems Networks and Digital Signal Processing*, pages 26–29, Newcastle, UK, July 2002. IEE/IEEE/BCS.

[5] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, December 1992.

[6] S. Mukkamala, A. H. Sung, and A. Abraham. Modeling intrusion detection systems using linear genetic programming approach. In *IEA/AIE'2004: Proceedings of the 17th international conference on Innovations in applied artificial intelligence*, pages 633–642. Springer Springer Verlag Inc, 2004.

[7] S. Sedaghat, J. Pieprzyk, and E. Vossough. On-the-fly web content integrity check boosts users' confidence. *Commun. ACM*, 45(11):33–37, 2002.

[8] D. Song, M. I. Heywood, and A. N. Zincir-Heywood. Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE Trans. Evolutionary Computation*, 9(3):225–239, 2005.

[9] T. Xia, G. Qu, S. Hariri, and M. Yousif. An efficient network intrusion detection method based on information theory and genetic algorithm. In *Performance, Computing, and Communications Conference, 2005. IPCCC 2005. 24th IEEE International*, pages 11–17, 2005.

[10] C. Yin, S. Tian, H. Huang, and J. He. Applying Genetic Programming to Evolve Learned Rules for Network Anomaly Detection. In *Advances in Natural Computation, First International Conference, ICNC 2005, Proceedings, Part III*, pages 323–331, 2005.

[11] Zone-H.org. Statistics on Web Server Attacks for 2005. Technical report, The Internet Termometer, 2006. Available at `http://www.zone-h.org/component/option,com_remository/Itemid,47/func,fileinfo/id,7771/`.