Automatic rule extraction from access rules using Genetic Programming

Paloma de las Cuevas, Pablo Garcia-Sanchez, Zaineb Chelly Dagdia, Maria-Isabel Garcia-Arenas, Juan Julian Merelo

Automatic rule extraction from access rules using Genetic Programming

Paloma de las Cuevas[1][0000−0002−5599−3981], Pablo García-Sánchez[2][0000−0003−4644−2894], Zaineb Chelly Dagdia[3,4][0000−0002−2551−6586], María-Isabel García-Arenas[1][0000−0002−5576−4417], and Juan Julián Merelo[1][0000−0001−8956−5304]

[1] Department of Computer Architecture and Computer Technology, ETSIIT and CITIC, University of Granada, Granada, Spain
{palomacd,mgarenas,jmerelo}@ugr.es
[2] Department of Computer Engineering, ESI, University of Cádiz, Spain
pablo.garciasanchez@uca.es
[3] Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
[4] LARODEC, Institut Supérieur de Gestion de Tunis, Tunisia
chelly.zaineb@gmail.com

Abstract. The security policy rules in companies are generally proposed by the Chief Security Officer (CSO), who must, for instance, select by hand which access events are allowed and which ones should be forbidden. In this work we propose a way to automatically obtain rules that generalise these single-event based rules using Genetic Programming (GP), which, besides, should be able to present them in an understandable way. Our GP-based system obtains good dataset coverage and small ratios of false positives and negatives in the simulation results over real data, after testing different fitness functions and configurations in the way of coding the individuals.

Keywords: Security · Corporate Security Policy · Genetic Programming · Rule Extraction · Bring Your Own Device.

1 Introduction and related work

In general, companies establish a series of rules to allow or reject access to assets from company-owned or bring-your-own devices (BYOD). These rules often depend on the context these devices are in and their specific characteristics. Although in general, asset access needs to be regulated, and the existence of devices not owned or controlled by the company adds a layer of complexity that makes the job of establishing an access policy more difficult. To this end, the Corporate Security Policies (CSPs) [12], approved by the company's Chief Security Officer (CSO), are the core at the identification of threats and the construction of a set of security rules. CSOs build the set of CSPs based on their expertise, and as such, in many cases access events are allowed or not depending on white- or blacklists or the simple presence or absence of a feature such as the fact that the asset is being accessed from a public, non-encrypted, WiFi.

The aim of this paper is to propose a novel technique for extracting inference rules from past behaviour instances that might help the CSO in the definition and refinement of security policies that, eventually, would classify an upcoming event or user action as permitted or not permitted. The objective, thus, is to obtain a way to classify correctly as many incoming events as possible, avoiding false positives [13], that is, avoiding the possibility of unsafe events being classified as safe.

Rules generated by GP could be used in two different scenarios. In the first one a CSO has hand-coded a set of security policies and wants to simplify or generalise them. The second scenario would simply dispense with rules and have the CSO manually decide which particular events are to be granted or denied access, and have a system such as the one described here generate a set of security policies by creating a set of rules from particular events. The main objective in both scenarios is to create a reliable rule set which is able to cover every new situation that might be a threat, allowing the system to go beyond the limited set of known pre-defined rules. Additionally, this feature can be used as “reverse engineering”, so that the rules initially made by the current or former CSO are found in the solution along with additional ones. In order to have a space of conceivable policy rules as wide as possible, it is necessary a technique that explores the rule space efficiently and with the least assumptions about rule structure.

This is why we have decided to use Genetic Programming (GP) for dealing with the problem of discovering novel, interesting knowledge and rules from large amounts of data [8], given that the up-to-date approaches are based in general on pre-defined or manually defined rules [1]. One of the advantages of GP is that by making the solutions to a problem available as trees, they themselves can be seen as decision tree classifiers [15] and can be expressed as a set of rules. Moreover, GP can outperform other methods such as SVM or naive-Bayes [4].

To the best of our knowledge, there is not a tool that helps CSOs in developing new security rules via GP, even as this method has been indeed applied to classification, as described by Espejo et al. in [6]. In fact, their survey theoretically supports our decision of applying GP to obtain security rules in a BYOD environment.

In our case, the assigned classes, or leaves of the tree, would be either “allow” or “deny”, acting over a certain incoming event; whilst the nodes are the conditions that have to be met to apply the action. Taking this into account, GP can be used to generate these classification trees, optimising an objective function called *fitness*. In this case the fitness can be defined as the accuracy of a rule or set of rules, being this the most used metric in classification [17], along with the classification error. But since there are other metrics that influence in “how good” a rule or a set of rules is, such as the depth of the created tree, the number of nodes it has, or the obtained false positives [3], it would be convenient to use them in the definition of the fitness.

The main issue we face is how to create a set of rules from a series of instances that those rules are bound to follow; initially we have a set of URIs and hand-coded tags that deny access to them, or allow it by default. Espejo et al. review in [6] three papers which use GP for classification with communications data, but mainly for intrusion detection and e-mail spamming. Thus, the works they review have applications in all fields but not exactly the one we focus on here. In [16] the authors also extract rules with IF ... THEN structure through GP, although for medical purposes. Furthermore, Alex A. Freitas deeply studied the application of GP to Data Mining (DM) in [8], providing the necessary knowledge and guidelines to design a GP framework for DM applications. Also, in [7], a system which discovers rules for the PROBEN1 databases [14], a collection of real-world datasets, via GP is described and a new fitness function is introduced. As happened in the survey of Espejo et al., out of six of the databases inside PROBEN1 and analysed by these authors, none is related to security. However, the authors' proposed fitness function is indeed of interest for this research, and as such we compare the performance or our algorithm using two different fitness ones: the more classical approach that measures only the correctly classified instances (accuracy); and De Falco's et al. [7] suggestion, also taking into account the complexity of the solution.

We thus demonstrate the utility of our framework by using it on a real-world dataset, and comparing two ways of coding the individuals – as a set of rules, or as a single rule – so that we are able to choose the best approach for our system. We make this comparison because while obtaining a set of rules as a solution is computationally expensive due to the need of longer evaluations, to present and evaluate a single rule not taking into account how it interacts with what the others cover [8] can lead to massive overlapping. Hence, we must study their accuracy despite of their advantages and disadvantages. In addition, we choose the most appropriate (fastest to converge and with best value) fitness after comparing the use of a most simpler one that only measures the accuracy, and a complex one which takes into account the complexity of the individuals. Finally, we propose the approach with the best performance in terms of best coverage over a validation set.

As previously highlighted, the main idea behind corporate security policies, which are defined by the CSO, is to build a basic, fixed, and well defined set of rules, which take the form of IF ... THEN clauses. By applying them, the company system decides if certain conditions are met in order to allow or deny access to an asset, being company owned or not, and wherever it is accessed from. Therefore, these rules can be visualised as the actions, taking place in a precise environment, being classified as allowed or denied. In this sense and while facing a security breach from a BYOD system, the set of rules will be tested looking for matches between the access characteristics and the rules premises – the conditions expressed in the IF part, also known as the description of the rule [7]. If it matches then the decision can be made, by checking the conclusion part of the rule, which comes after the THEN and indicates the class [7], either by allowing or denying employees' access to non-confidential or non-certified data, for example. However, it is important to mention that the companies' set of security rules defined by the CSO is based on known and previously recognised accesses and thus it cannot cover the whole, safe and unsafe, search spaces. Therefore, there is an urgent need to develop a system capable of discovering a more reliable rule set which should be able to cover every new situation that may be a threat. Hence, allowing the company security system to go beyond the limited set of known, pre-defined rules.

The rest of the paper is organised as follows. Next (Section 2) describes the proposed methodology, depicting the problem this work tries to solve and describing the available dataset and the proposed GP framework. The experimental setup, as well as the different set of experiments that have been carried out are also described in that section. Section 3 shows and discusses the obtained results from the application of GP to security rules extraction and, finally, the conclusions of this work along with some suggestions about how to continue our research are given in Section 4.

2 Methodology

Our proposed solution is based on a novel GP framework dedicated for the BYOD context and capable of performing an automatic and wider discovery of classification rules. More precisely, our GP based framework will, first, extract all the possible values of every attribute in the data at hand and then make the GP algorithm evolving. Specifically, in this context, we have decided to follow the more conventional approach in Genetic Programming, the Pittsburgh approach [8], meaning that each individual is seen as a set of rules. However, in this work we have also implemented the Michigan approach, where every individual is a single rule. The aim of having these two different implementations is to choose the most efficient, in terms of time to find the solution, best fitness, accuracy in the validation phase, and readability.

The last step would be to present the rules – solution – to the CSO of the company and tune up the algorithm according to the decision of finally including or not the set of rules in the main security policy. The description of the used data and further explicit details about our proposed solution are given next.

The set of used data has been gathered from the evaluations that were performed during the development of an FP7 European Project, called Anon. In these evaluations, a group of users tested a smartphone and PC *app* meant for securing a BYOD environment. The app generates warnings when the users act in a dangerous way. Technically, these warnings are triggered by a set of initial and pre-defined rules. When certain conditions are met in an "event" (action performed by a user), the corresponding action could be allowed - nothing happened - or denied with a warning explaining the rule that the user did not comply with. Then, the app displays the steps to perform the action in a more secure way or environment.

The dataset contains a collection of these "events" from which a number of attributes (variables) have been extracted or were given by the application itself.

User data has been also extracted but anonymised, in the sense that from all the attributes extracted from the user actions, those that could lead to identifying the user are not included as variables to build rules with. The attributes can be classified in different ways; one of them is based on whether they are directly read from the application or inferred after processing the data. Therefore, we distinguish between:

- Attributes given by the tested application: these attributes are related to the type of the event (action), its timestamp, or the application which originated the event, among others.
- Attributes inferred from the information in the database: the information given by the aforementioned attributes, along with the rest of information already existing in the database, helps inferring other attributes. These are, for instance: all extra information related to the origin, like the user position in the company or the device Operating System; the configuration of the device, such as WiFi or Bluetooth being enabled; and even lexical properties of the user password, in order to avoid storing the password itself or using it for classification or rule generation.

The tests had a duration of five weeks, and a total of 153270 events were registered in the database. We discarded those events that did not imply access to assets, meaning that they were not useful for knowledge extraction purposes, such as events of *log in*, *log out*, or *restarting the server*. The remaining was a 35% (53296 instances) of the total, and were considered as *important* because they contain information about meaningful user actions such as opening files or sending emails in a certain connection environment, changing security properties, or installing apps. Altogether, there are 38 attributes, plus the class, which can take two possible values: GRANTED or STRONGDENY.

As previously highlighted, in this work we propose a system which is able to process a set of user actions that have been allowed or denied based on initial, simple rules, and discover new rules through GP by exploring the whole space of possible combinations among the attribute values. The coding of the individual might take two approaches, named *Pittsburgh* and *Michigan* [8].

The Pittsburgh approach uses GP to create an individual tree that models a set of different rules, given that the problem can be seen as a classification one and therefore the model can be a decision tree [15]. Then, the generated tree is a binary tree of expressions formed by two different types of nodes:

- *Variable*: it is a logical expression formed by a prefix, a name, an operator and a value. It is the equivalent to a "primitive" in the field of GP [3]. The operators depend on the type of variable, being {=>} (an arrow, as in "takes the value of") in the case of categorical attributes, {=} for binary attributes, and {<, <=,=,=>, >} for numeric ones. At the same time, the prefix can be {AND, OR}, and NOT can appear before these.

  $$\text{Examples: }\begin{cases}\texttt{password\_length<5}\\ \textit{or}\\ \texttt{event\_level=>COMPLEX\_EVENT}\end{cases}$$

- *Action*: it is a leaf of the tree and therefore, a "terminal" state. Each decision is the result of applying the rule, so it is limited to two terms which are `GRANTED` or `STRONGDENY`. Only one leaf must hung from a parent (variable) node.

Rules are constructed starting from each leaf of the tree and iterating to the upper parents, or variables, reading their data as string. Therefore, the number of rules of the set produced by the tree is equal to the number of its leafs. It is worth mentioning that some rules might have contradictory conditions inside them during the evolution. This is not a problem because those rules will not cover any instance and thus they will not contribute to the fitness value.

The second approach tested, called Michigan approach, assigns a single rule to every individual. In this paper we have expressed the rule as a list of conditions, with a fixed class, obtaining just one rule per execution. That means we are not using GP in this case, because the generated individual is not a tree, but a vector, so we are applying a regular Genetic Algorithm (GA) instead. Therefore, in order to cover all classes, the algorithm has to be executed once for each class; in our case, GRANTED, allowed actions, and STRONGDENY, for the denied actions.

Indeed, each approach has its advantages and disadvantages. The Pittsburgh approach allows to directly obtain a set of rules able to classify instances of every existing class, meanwhile Michigan approach solution is coded as a single rule, so that we obtain as many rules as classes are defined. The possibility of having many rules for every class instead of just one, more general, per class might seem to better help the CSO in detecting specific dangerous situations. At the same time, obtaining a set of rules as a solution is more computationally expensive due to the need of longer evaluations. Lastly, to evaluate a single rule not taking into account how it interacts with what the others cover [8] can lead to massive overlapping with the consequent loss of efficiency.

With respect to the variables, both approaches use three different types, described as follows [17]:

- *Binary Variable*: those with a boolean value, for instance, variables that are related to the device services switched on or off and important features such as the device having or not an antivirus installed.
- *Categorical Variable*: the ones with nominal values, where a list is defined with the possible values it may have, in order to randomly pick up one in the creation of the rules.
- *Numerical Variable*: those with a numerical value, for which both maximum and minimum values are specified.

We distinguish the variables used by the GP expressions in order to create the rules in those that are specific to the BYOD context and those that will show up in any environment:

BYOD-specific : DeviceHasAccessibility, DeviceHasAntivirus, DeviceHasPassword, DeviceIsRooted, DeviceOS, DeviceOwnedBy, WifiEncryption, DeviceScreenTimeout, PasswordLength

General BluetoothConnected, MailHasAttachment, WifiConnected, WifiEnabled, AssetConfidentialLevel, DeviceType, EventLevel, EventType, UserRole

There is no difference, however, in how do we deal with the two types of variables; this only shows the exhaustiveness of the set of terminals that we will be using in this work. At the end of the process, either as part of a set of rules or just being a single rule, they can be presented as follows:

```
device_has_antivirus=false AND
password_length<5 AND
user_role=>Administration OR
device_is_rooted=true THEN=STRONGDENY
```

Rules presented in this way offer good readability which is key to understand the relationship between attributes and how the described situation might, or might not, be dangerous. In the example, the system would have inferred that an action from a device without antivirus, a short password, and rooted or belonging to an administration employee, should be denied.

In the application of GP to classification the most used metric to evaluate the individuals, i.e. the fitness function, is the accuracy [6]. The accuracy is normally obtained as the ratio of the correctly classified instances among the total of instances. Witten et al. use in [17] the concepts of true positive/negative and false positive/negative. The first refers to the correctly classified instances, and the latter are those instances that are classified as the contrary, and consequently they are called false positives or false negatives. Using this nomenclature – true negative (TN) and true positive (TP) –, a fitness function defined for accuracy would be expressed as follows:

$$f_{Acc} = (TP + TN)/T_{tr} \tag{1}$$

This kind of fitness has to be maximised, given that the ideal value of f_{Acc} is the whole training dataset, T_{tr}. Equation 1 has the advantage of not being computationally expensive, but it does not penalise the badly predicted instances (false positives and negatives), which in security environments such as this one can be very harmful. Furthermore, a false negative would mean that the system denies an event that should be allowed, but the worst-case scenario is having a false positive, when a dangerous event is classified as allowed. To this end, in [17] the authors define the coverage as:

$$C_{ind} = TP + TN - (FP + FN) \tag{2}$$

Additionally, to take into account the complexity of the individuals, whether they are a rule or a set of rules, in [17] they introduce a measure of the generated trees or list size by this expression:

$$S_{ind} = N_{nodes} + depth \tag{3}$$

Where N_{nodes} is the number of nodes of the tree (or elements in a list), and *depth* is the tree depth. So that combining C_{ind}, S_{ind} and introducing an α

variable inside [0, 1] to tune up the degree of allowed complexity, the problem becomes now a matter of minimising this formula:

$$f_{CS} = (T_{tr} - C_{ind}) + \alpha S_{ind} \tag{4}$$

Therefore, in our experimental section we are going to compare both f_{Acc} and f_{CS}, with different alpha values, to check two statements:

- How do they influence in the number of evaluations taken to find the best solution.
- Which fitness function is able to minimise false positives and negatives.

And finally decide which fitness function offers, taking into account these measures, the best performance.

Once the methods to compare have been explained, the rest of the experimental setup is now described.

The configurations that will be compared involve two different encoding of individuals (Pittsburgh tree individuals vs. Michigan list individuals), two types of fitness (f_{CS} and f_{Acc}), and three different values for α in the case of f_{Acc}.

With respect to the GP parameters, different decisions for experimental design have been taken into account. First, sub-tree crossover and 1-node mutation evolutionary operators have been used, as indicated by, for instance, [9]. In this case, during the mutation operation, there is a 50% of probability to change the complete variable (prefix, name, operator, and value) or only the value. A population of 32 individuals and a 2-tournament selector for a pool of 16 parents have been used. These parameters have been also previously used in, for instance, [9]. Table 1 summarises all the parameters used.

Table 1. Parameters used in the experiments.

Parameter Name	*Value*
Population size	32
Crossover type	Sub-tree crossover
Crossover rate	0.5
Mutation	1-node mutation
Selection	2-tournament
Replacement	Generational with elitism
Stopping criterion	150 generations
Maximum Tree Depth	10
Runs per configuration	10
Compared configurations	
Individual representation	Pittsburgh vs. Michigan
Fitness	f_{CS} VS f_{Acc}
α for f_{CS}	0, 0.5, and 1

During the fitness evaluation, the generated individual is transformed into a string, which can become a single rule or set of rules depending on the approach, as previously described. Then, the chosen fitness is evaluated for a particular rule – the single rule or each one inside the set – and over the 90% of the data. For further reliability of the results, it is advised to perform 10-fold cross-validation, and as such the WEKA Java Library [10] has been used to generate the 10 folds or distributions of data into 90% training (fitness evaluation) and 10% validation. This way, each experiment has been executed 10 times, each one with a different distribution of data.

The algorithms have been executed in a cluster node with 16 Intel(R) Xeon(R) CPU E5520 @2.27GHz processors, 16 GB RAM, CentOS 6.8 and Java Version 1.8.0_80. The specific source code of the proposed method is available under a LGPL V3 License at `github.com/anon` , as a module for the framework Anonymous n

3 Results from Genetic Programming application

As our purpose in this paper is to obtain a system which proposes to a CSO useful security rules that improve the existing ones, and present them in an understandable way, in this section we compare the results from using two fitness functions described in Equations 1 and 4, as well as the two approaches taken – Pittsburgh and Michigan –. In addition, an example of the individuals obtained will be presented in order to understand how the different approaches affect them, along with their advantages and disadvantages. Lastly, we will choose the best approach, justifying this choice.

The aim of this comparison is to conclude which fitness function should be used, discussing the results from the point of view of convergence, and that means finding which one reaches the best value faster. To study this we have displayed the obtained fitness through the iterations for the Pittsburgh approach in Figure 1 and for the Michigan approach in Figures 2 and 3. Figures for GRANTED and STRONGDENY classes are separated because the solution is a single rule instead of a set of rules, and therefore the algorithm has to be executed once per class. With regard to the best fitness obtained for each fitness function, they are similar in the same scope, which means that the different values of α in f_{CS} did not present significant differences in the two approaches separately.

For the sake of clarity, f_{CS} has been divided by the maximum value – the number of instances for training – it might take. By looking at the Pittsburgh approach values in Figure 1, we show that mostly all configurations tend to converge around iteration 40, but it seems that f_{CS} with $\alpha = 0.5$ is the configuration that reaches the best solution faster, around the 30th iteration.

With respect to the Michigan approach, a higher variability is noticeable in Figures 2 and 3, but in average we see that for f_{CS} with $\alpha = 0$ or 1, the fitness do not converge until generation 120. The best ones are for f_{Acc} and f_{CS} with $\alpha = 0.5$, being the latter the one that converges faster, around the 50th iteration.

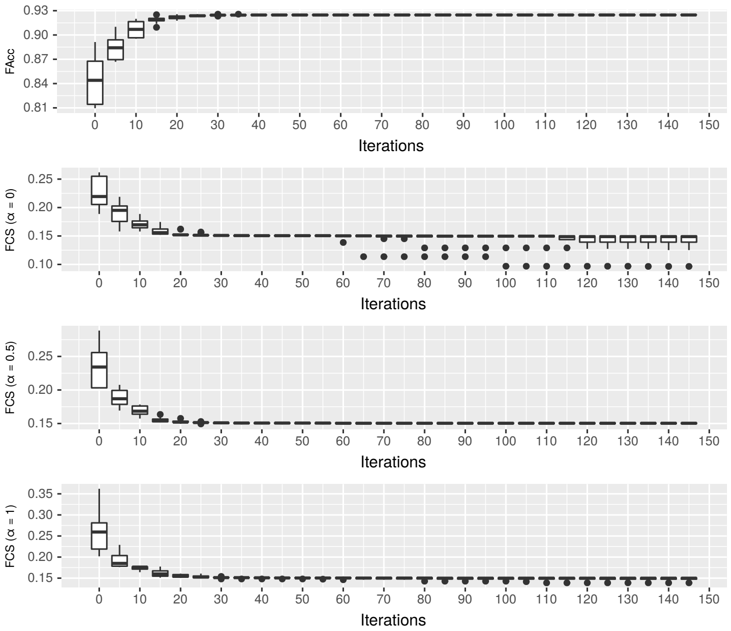


Fig. 1. Convergence of fitness for each one of the tested fitness functions, following the Pittsburgh approach. Note that f_{Acc} has to be maximised, whereas f_{CS} has to be minimised.

We can advance that the best results might be obtained for f_{CS} with $\alpha = 0.5$. However, there is a considerable difference between the performance, in terms of best obtained fitness, of the two used approaches. In this way, we have to thoroughly compare them. In the next section we do so by choosing the validation coverage and the ratio of false positives and negatives as independent measurements.

Once the variability of the obtained fitness has been studied, and always taking into account that those values come from their evaluation in a training subset of the data, now we *validate* the proposed approaches with a validation set, similar to the validation set used in classification problems [17]. The way to evaluate this is similar to Equation 1, but using the validation subset of the data:

$$v_{Acc} = (TP + TN)/T_{val} \tag{5}$$

This measure will be the same independently of the approach or fitness function has used to evaluate the individuals. Table 2 shows the average, best, median, and worst results from the evaluations with f_{Acc} and also f_{CS}.

In Section 3 we have shown that the performance in the fitness of the Michigan approach was worse than that of Pittsburgh, and now it is more clear, given that the accuracy over the validation set is not even 50%. In fact, the dataset

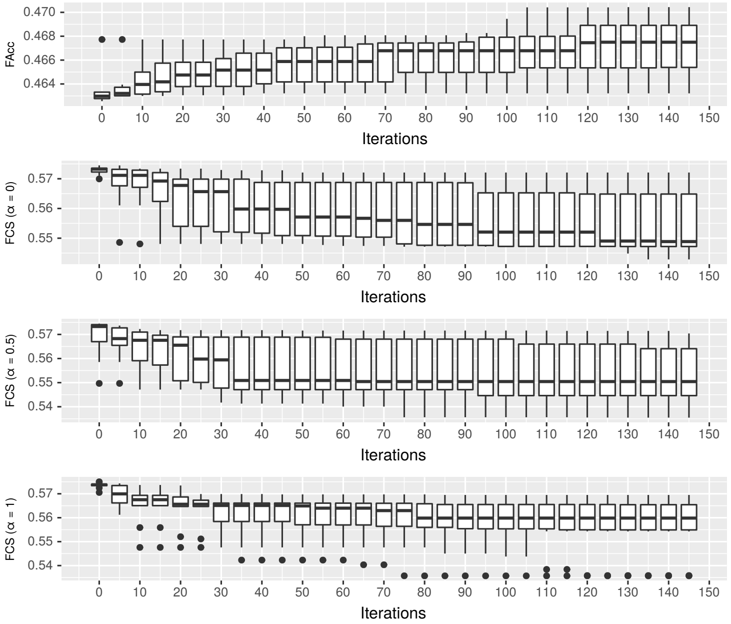


Fig. 2. Evolution of fitness for the Michigan approach and the GRANTED class.

Table 2. Validation (accuracy in classification of a new dataset) scores for the Pittsburgh and Michigan approaches. $*$ marks statistically significant best values.

				Validation measurement					
		Fitness function		Average	Best	Median	Worst	FP	FN
Pittsburgh: Individual eq. set of rules		f_{Acc}		0.925 ± 0.379e-2	0.929	0.926	0.917	0.075 ± 0.045e-1	0
		f_{CS}	$\alpha = 0$	0.926 ± 0.744e-2	0.945	0.926	0.917	0.072 ± 1.272e-2	0.002 ± 0.564e-2
			$\alpha = 0.5$	0.924 ± 0.371e-2	0.929	0.924	0.917	0.074 ± 0.451e-2	0
			$\alpha = 1$	0.925 ± 0.384e-2	0.929	0.926	0.917	0.075 ± 0.378e-2	0
Michigan: Individual eq. one rule	Class: GRANTED	f_{Acc}		0.467 ± 0.143e-2	0.472	0.468	0.459	0.023 ± 0.095e-1	0
		f_{CS}	$\alpha = 0$	0.466 ± 0.441e-2	0.472	0.467	0.459	0.021 ± 0.094e-1	0
			$\alpha = 0.5$	0.467 ± 0.305e-2	0.472	0.467	0.462	0.019 ± 0.088e-1	0
			$\alpha = 1$	0.465 ± 0.43e-2	0.472	0.467	0.458	0.023 ± 0.099e-1	0
	Class: STRONGDENY	f_{Acc}		0.046$*$ ± 0.74e-2	0.065	0.045	0.037	0	0.307 ± 0.745e-1
		f_{CS}	$\alpha = 0$	0.025 ± 1.527e-2	0.045	0.023	0.003	0	0.004 ± 0.039e-1
			$\alpha = 0.5$	0.02 ± 1.603e-2	0.061	0.015	0.002	0	0.007 ± 0.063e-1
			$\alpha = 1$	0.025 ± 1.948e-2	0.047	0.036	0	0	0.004 ± 0.049e-1

we are using is highly imbalanced; there are 1 instance in the training set of data labelled as STRONGDENY for every 13 labelled as GRANTED. Thus, the results we have obtained are biased towards the majority class [11].

At the same time, and because of the distributions for the Michigan approach follow the normal, an ANOVA test has been performed in every class [5], obtaining a p-value of 0.5538 for the GRANTED class, meaning that there are not statistically significant differences in the results. However, in the case of the STRONGDENY class, using f_{CS} significantly (p-value of 0.001736) decreases the accuracy over the validation set. With regard to how many FP and FN are

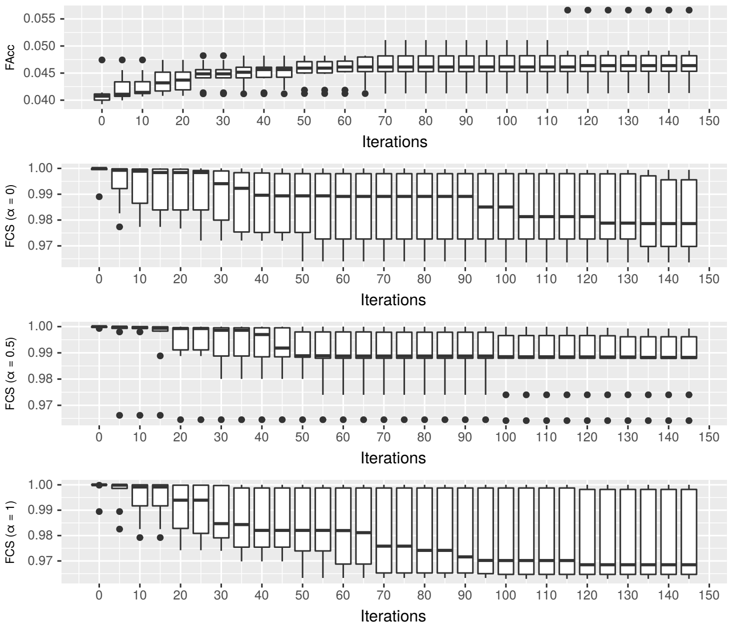


Fig. 3. Evolution of fitness for the Michigan approach and the STRONGDENY class.

obtained in every approach, we see in Table 2 that for the Pittsburgh approach we generally obtain low rates or 0 – ideal – false negatives, and around 7% rate of false positives – almost 400 in average for our validation set –. On the other hand, best rates are found for the Michigan approach, where the rate of FP is, at most, 2.8% – around 113 instances in average –. And even if there were FN found, the average for the used validation set is 29 instances out of 5330, which is a very low number and as we explained before, in this BYOD scenario it is less worse to have FN than FP. Also, this imbalance in the FP and FN values is also caused by the imbalance in the dataset.

To evaluate computation costs of the two approaches, and taking into consideration the infrastructure available for the experiments (see Section 2), we present the execution times in Table 3, detailed by their average value and standard deviation, along with the best, worst, and median values. Time is expressed in hours, for the sake of clarification. In addition, the values for the Michigan approach are not separated by class this time, because in order to have the two rules - one for each class – the CSO would have to wait for both executions.

In this table we see that best times are obtained when f_{CS} is used. More precisely, in the Pittsburgh approach, the distributions follow the normal and after performing the ANOVA test, we can say that the shorter execution time (the best) is found for $\alpha = 0.5$ with statistical significance (p-value of 0.008623). In the case of the Michigan approach, we again find statistical differences (p-value of 5.537e-05) between the results, and thus we can say that the best execution

Table 3. Execution time, Pittsburgh and Michigan approaches; this one adds times for GRANTED and STRONGDENY. $\ast$ marks statistically significant best values.

			Time measurement (h)			
	Fitness function		Average	Best	Median	Worst
Pittsburgh: Individual eq. set of rules	f_{Acc}		18.371 ± 7.178	8.545	17.366	31.366
	f_{CS}	$\alpha = 0$	14.424 ± 2.938	7.016	14.352	17.43
		$\alpha = 0.5\ast$	11.934 ± 2.22	8.267	11.609	16.166
		$\alpha = 1$	13.176 ± 1.923	9.283	13.424	16.517
Michigan: Individual eq. one rule	f_{Acc}		10.627 ± 0.187	10.307	10.64	10.956
	f_{CS}	$\alpha = 0\ast$	9.342 ± 0.3	8.993	9.261	9.854
		$\alpha = 0.5$	9.435 ± 0.348	9.039	9.401	10.281
		$\alpha = 1$	9.612 ± 0.508	8.946	9.509	10.38

time is obtained when we use f_{CS} with an α value of 0. It seems that the time does decrease when α is different from 0, meaning that the depth of the tree has influence on the evaluation of the fitness.

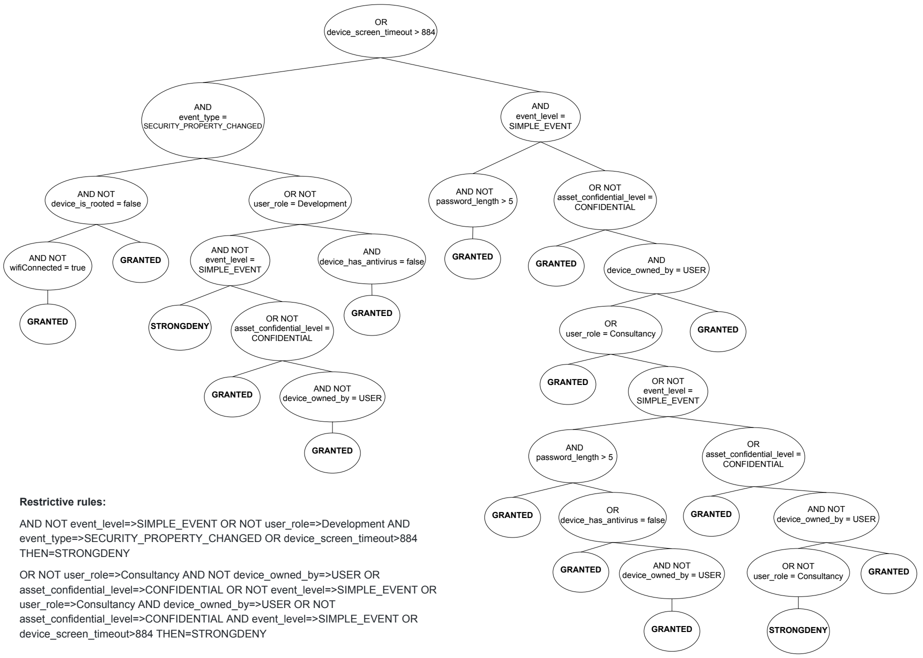
To study this effect, we have to look at the sizes of the best individuals obtained for each configuration, which are displayed in Table 4.

Table 4. Tree size of the best individuals for the Pittsburgh approach. An $\ast$ indicates the statistically significant best value for α.

			Best individual size			
	Fitness function		Average	Best	Median	Worst
Pittsburgh: Individual eq. set of rules	f_{Acc}		60.2 ± 10.922	47	60	79
	f_{CS}	$\alpha = 0$	60 ± 11.086	43	60	83
		$\alpha = 0.5$	40 ± 4.447	35	40	47
		$\alpha = 1\ast$	36.2 ± 3.011	31	37	39

In this table we only show the results for the Pittsburgh approach, given that the sizes of the trees do change in GP, but the size of a list – number of conditions in the rule – does not, and in this case is always 10. Certainly, the size of the trees decreases when the value of α grows. Since the distributions do not follow the normal, a non-parametric test (Kruskal-Wallis) has been used to asset the statistical significance [5], obtaining a p-value of 1.679e-10. Then, the smaller individual is found for f_{CS}, and $\alpha = 1$. In the context of this problem, a smaller number of rules increases their explainability, which is why this particular value and fitness function will have to be considered if that is a priority.

What we also have to determine is which kind of presented solution is better: showing the CSO a set of rules or a single rule for each class. Figure 4 shows an example of best individual obtained using f_{CS} and $\alpha = 0$. The tree in this figure represents a set of 16 rules, two of them classifying to the STRONGDENY class, whilst the rest classify towards the GRANTED class. The ones that classify to STRONGDENY have been highlighted and presented in string form in the figure.



Restrictive rules:

AND NOT event_level=>SIMPLE_EVENT OR NOT user_role=>Development AND event_type=>SECURITY_PROPERTY_CHANGED OR device_screen_timeout>884 THEN=STRONGDENY

OR NOT user_role=>Consultancy AND NOT device_owned_by=>USER OR asset_confidential_level=>CONFIDENTIAL OR NOT event_level=>SIMPLE_EVENT OR user_role=>Consultancy AND device_owned_by=>USER OR NOT asset_confidential_level=>CONFIDENTIAL AND event_level=>SIMPLE_EVENT OR device_screen_timeout>884 THEN=STRONGDENY

Fig. 4. Example with f_{CS} and $\alpha = 0$.

The STRONGDENY rules in Figure 4 imply that the developers are more probably to get a GRANTED in their actions, and this conclusion makes sense because they are supposed to be more familiar with systems security. Also, from the second rule we can infer that the complex events – not simple – and the BYOD devices imply a certain degree of unsafety. The root, and thus most important, condition, is related to the time that the the device takes to enter into sleep mode (screen turns off).

4 Conclusions and Future Work

In this work, we wanted to help companies adopting the BYOD philosophy by developing a tool that is able to discover rules using a GP approach, extracting knowledge from the users behaviours when interacting with their devices, at the time it minimises the false positives, so that a dangerous action is never taken as permitted. To this end, in Section 2 we have presented different ways to implement a methodology based on GP, by two approaches: Pittsburgh, in which the individuals are coded as set or rules; and Michigan, where the individuals are a single rule, and thus a rule for every class has to be generated. In addition, two different fitness functions have been used to study their effect over the validation accuracy rate and the number of false positives and negatives.

The results in Section 3 make a fine proof of concept of a tool that helps the CSO of a company discovering dangerous situations through the presentation of GP generated rules. We obtained promising rates for the number of false positives (around 7%), and good coverage over the validation data set, as is shown in Section 3. However, since the ideal value for both false positives and negatives is 0, there is room for improvement. In addition, Figure 4 shows that the individuals are presented in a readable form, but the ratio between rules classifying to one or another class is too biased towards the GRANTED class.

With regard to the execution time, the solution we propose takes between 16 and 17 hours, in the worst case and taking into account the used infrastructure, to obtain the best individual. Although these times could be acceptable, in the sense that the CSO would have new rules every day, they can still be reduced. A possible way to achieve better values for FP and FN is to set up the problem as a multiobjective one, so that we try to minimise f_{CS}, but also FP and FN. Additionally, we will try to reduce the execution time by adapting the algorithm parameters, for instance, the number of generations, as has been shown in Section 3 that in some cases 50 generations is enough. Other enhancements could be to study every tree individual and not to directly evaluate the fitness for the contradictory rules of each one.

On the other hand, the imbalance of the dataset should be taken into account, given that it affects the results in terms of obtained accuracy in the validation process and the ratio among classes in the obtained individuals [2], so different solutions to deal with this issue will be applied.

For our future work we will implement these solutions to continue improving our system. Furthermore, we will extend the approach to other problems where data is available. For example, particularisations of the BYOD problem as it could be Internet navigation during work hours. We will explore network traffic from public data repositories, such as `http://www.secrepo.com/#3p_network`, and try to apply our approach.

5 Acknowledgements

Acks taking
this much space.

References

1. Ali, S., Qureshi, M.N., Abbasi, A.G.: Analysis of BYOD security frameworks. In: 2015 Conference on Information Assurance and Cyber Security (CIACS). pp. 56–61. IEEE (2015)
2. de Arruda Pereira, M., Carrano, E.G., Junior, C.A.D., de Vasconcelos, J.A.: A comparative study of optimization models in genetic programming-based rule extraction problems. Soft Comput. **23**(4), 1179–1197 (2019)
3. Back, T.: Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford university press (1996)

4. Castellanos-Garzón, J.A., Ramos, J., Martín, Y.M., Paz, J.F.D., Costa, E.: A genetic programming approach applied to feature selection from medical data. In: Fdez-Riverola, F., Mohamad, M.S., Rocha, M., Paz, J.F.D., González, P. (eds.) Practical Applications of Computational Biology and Bioinformatics, 12th International Conference, PACBB 2018, Toledo, Spain, 20-22 May, 2018. Advances in Intelligent Systems and Computing, vol. 803, pp. 200–207. Springer (2019)
5. Derrac, J., García, S., Molina, D., Herrera, F.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation **1**(1), 3–18 (2011). https://doi.org/10.1016/j.swevo.2011.02.002, `http://dx.doi.org/10.1016/j.swevo.2011.02.002`
6. Espejo, P.G., Ventura, S., Herrera, F.: A survey on the application of genetic programming to classification. IEEE Transactions on Systems, Man, and Cybernetics, Part C **40**(2), 121–144 (2010)
7. Falco, I.D., Cioppa, A.D., Tarantino, E.: Discovering interesting classification rules with genetic programming. Applied Soft Computing **1**(4), 257–269 (2002). https://doi.org/http://dx.doi.org/10.1016/S1568-4946(01)00024-2, `http://www.sciencedirect.com/science/article/pii/S1568494601000242`
8. Freitas, A.A.: Data mining and knowledge discovery with evolutionary algorithms. Springer Science & Business Media (2002)
9. García-Sánchez, P., Fernández-Ares, A., Mora, A.M., Valdivieso, P.A.C., González, J., Guervós, J.J.M.: Tree depth influence in genetic programming for generation of competitive agents for RTS games. In: Esparcia-Alcázar, A.I., Mora, A.M. (eds.) Applications of Evolutionary Computation - 17th European Conference, EvoApplications 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8602, pp. 411–421. Springer (2014). https://doi.org/10.1007/978-3-662-45523-4_34
10. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: An update. SIGKDD Explorations **11**(1) (2009)
11. Japkowicz, N., Stephen, S.: The class imbalance problem: A systematic study. Intelligent data analysis **6**(5), 429–449 (2002)
12. Kaeo, M.: Designing Network Security, Second Edition. Cisco Press (2003)
13. Pietraszek, T., Tanner, A.: Data mining and machine learning - towards reducing false positives in intrusion detection. Information security technical report **10**(3), 169–183 (2005)
14. Prechelt, L.: PROBEN 1-a set of benchmarks and benchmarking rules for neural network training algorithms (1994)
15. Safavian, S.R., Landgrebe, D.: A survey of decision tree classifier methodology. IEEE Transactions on Systems, Man, and Cybernetics **21**(3), 660–674 (May 1991). https://doi.org/10.1109/21.97458
16. Tsakonas, A., Dounias, G., Jantzen, J., Axer, H., Bjerregaard, B., von Keyserlingk, D.G.: Evolving rule-based systems in two medical domains using genetic programming. Artificial Intelligence in Medicine **32**(3), 195–216 (2004). https://doi.org/http://dx.doi.org/10.1016/j.artmed.2004.02.007, `http://www.sciencedirect.com/science/article/pii/S0933365704001058`, adaptive Systems and Hybrid Computational Intelligence in Medicine
17. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann (2005)