request) whether the call should be challenged or not. The performance of the policy is evaluated through the feed-back of the end-users. Real-time and adaptive refinement of the protection policy requires machine learning techniques that are both: (1) adaptive: face to the continuous change of both the attack strategies and the legal activities, and (2) expressive: i.e. defines concrete signatures. In particular, the decision trees are known for their high expressivity but they support only slight changes over time. Although proposals for adaptive decision trees exist ([10, 2]), we propose a more appropriate approach similarly to [3]: we extract signatures from decision trees learnt in parallel then the signatures are combined with conflict resolution.

The rest of this paper is organized as follows. In section 2, we summarize the problem statement and expose the SIP firewall configuration framework. Section 3 presents the decision trees construction, signature extraction and updating. In section 4 the experimentation and results are exposed. Section 5 reviews the related works. In section 6 we conclude the paper and mention future works.

2. SIP FIREWALL CONFIGURATION FRAMEWORK

The goal of the proposed framework is to protect the VoIP subscribers from incoming SPIT calls. The problem statement can be summarized in the following points:

- The identification of SPIT should be prior to answering the call, hence no information about the call content is available.
- SPIT can be generated by a botnet, that is spread over hundred or thousands of IP addresses and blacklisting does not scale.
- SPIT calls can be routed together with legal calls through the same VoIP peer network, that is blocking all the traffic incoming from that peer is not feasible.
- Applying active tests for identifying SPIT contributes to the call setup delay and consumes server resources. Only a small subset of incoming calls should undergo these tests.

The only information available previously to routing the call is the content of the SIP INVITE message. We define the signature of a group of INVITE messages as a logical expression of sentences where each sentence is composed of a feature, a value and a logical operator. This logical expression needs to be true for all the SIP messages it represents. A feature can be any field in any SIP or SDP[1] header as shown in Figure 1. Other features may reflect the count or the position of a field, for example, the number of Via headers, or the position of the Call-ID within the message. Fields from routing and transport layers can also be used (i.e. the IP source, the transport protocol).

The strength of our approach comes from the diversity of SIP user agents, servers, configurations and implementations in the market which results in differently structured and assigned INVITE messages. The attackers have only partial information on the content of the benign messages

[1]SDP: Session Description protocol is coupled with SIP and is responsible of media negotiation[5]

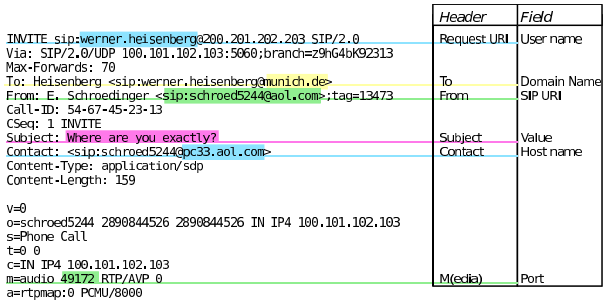

Figure 1: Example of a SIP message and possible features

and therefore can not simulate their signatures. Our framework for mitigating the SPIT threat is shown in Figure 2 and consists in the following components:

A **SIP firewall** has for mission to inspect the incoming INVITE messages based on the protection policy. A classifier recognizes the SPIT signatures and challenge the callers by a Captcha-like test before forwarding the calls. In case the challenge system is flooded with SIP INVITES known to trigger the classifier, a rate-limiting component directly drops the call requests.

The end user-agent is equipped with an **exporter** for labeling received calls. For example, after the call hang-up the user presses a button or indicates in a pop-up window (for softphones) that he received a SPIT. This is usual in IP telephony since many providers pops-up a window for a satisfaction query about the quality of the last call. If no button is pressed, the exporter labels the message as benign. A particular type of end-users would be a honeypot user agent where every received call is considered as SPIT. For this paper, we assume all the user feedback is trustworthy. We plan to incorporate a trust model in future works.

The **collector** matches the spit/benign information given by the exporters and the challenged/not challenged information given by the firewall. This information is used to provide a labeled training data-set for the supervised learning algorithm.

The supervised **machine learning** component is responsible of processing the labeled data and generating the benign and SPIT signatures. The conflicts with old signatures are then resolved and the protection policy is updated. The protection policy has three kind of rules: SPIT rules apply when a SPIT signature is recognized, normal rules apply when a normal signature is recognized and default rules apply when neither is found.

3. DECISION TREES

Decision trees are a widely used class of methods for learning and classification. The instances presented as arguments for these methods are represented as collections of attribute-value pairs. A decision tree represents a disjunction of conjunctions of constraints on the attribute values of the instances. Each node from the tree represents a test regarding the value of an attribute, each path starting from the root node to a leaf node represents a conjunction of attribute tests and the tree synthesizes the disjunction of these conjunctions.

Our problem is reduced to decision trees with categorical

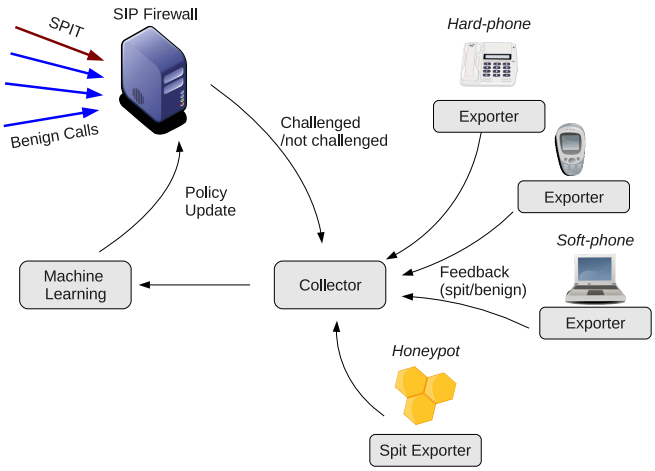

Figure 2: A Framework for Self-Protection against SPIT

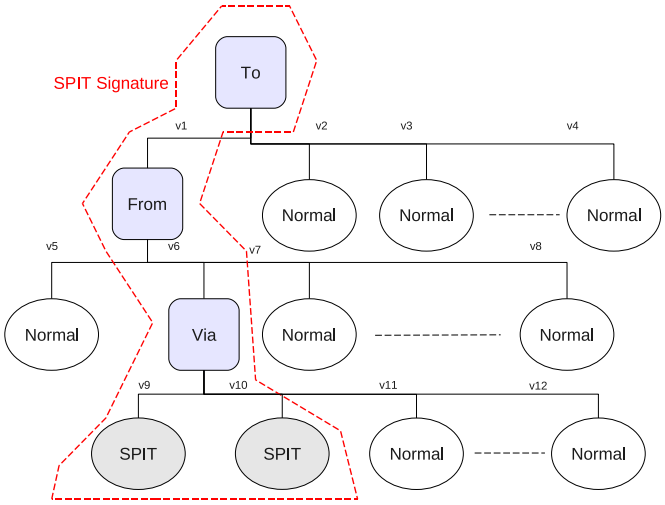

Figure 3: Extracting a SPIT Signature from a Decision Tree

attributes where the values of each attribute are the string tokens extracted from the SIP messages (i.e. the features). Since the different categories of a string attribute are practically infinite, we limit the categories to those we found in the current training data-set.

The learning of a decision tree starts by the question: "What constraint should be tested in the root of the decision tree?". Then it evaluates the available constraints in accordance with a statistical test to select the best one. A descendant of the root node is created for each value of this constraint and the entire process is repeated for each descendant. The two most used statistical tests are the information entropy and the Gini index. The C4.5 tree generation algorithm [12] is based on the information gain. The information gain is the difference of two entropy quantities:

- $info(L)$ is the entropy of the set L relative to the k-wise classification (in our case k=2: Spit and benign), that is before making any split.
- $info_T(L)$ is the weighted sum of the entropies over the n subsets obtained by applying a test T (i.e. choosing an attribute for splitting the data).

$$gain(T) = -\sum_{i=1}^{k} p(c_i) log_2(p(c_i)) + \sum_{i=1}^{n} p(t_i) \sum_{j=1}^{k} p(c_j/t_i) log_2(p(c_j/t_i))$$

The information gain criterion selects a test that maximizes the information gain function. We use J48 which is an open source Java implementation of the C4.5 algorithm from the WEKA machine learning library [4].

3.1 Signature Extraction

The next step is to extract the signatures from the learnt decision tree. In Figure 3, we have this SPIT signature:

```
To=v1 and From=v6 and (Via = v9 or Via= v10)
```

The signature extraction is based on the following procedure:

1. For each leaf node, we identify the class ("spit" or "normal") and the data instances it represents.
2. At each level of the tree, we group the leaf nodes into two groups based on their classes. We ignore all the leaf nodes that do not represent any data instance.
3. We calculate the entropy for each of the two groups: $(-\sum_{i \in categories} p(i) \log_2 p(i))$.
4. We generalise over the class of the group having the largest entropy.
5. The tree is then translated into if-else rules and the conditions leading to SPIT form the signature.

For the example in Figure 3[2], this procedure gives:

```
if (To==v1):
        if (From==v2):
                if (Via== v9 or Via ==v10):
                        return "spit"
                else:
                        return "normal"
        else:
                return "normal"
else:
        return "normal"
```

The procedure may lead to extract normal signatures as well. The generalisation heuristic (step 4) helps configuring the default rules in the protection policy. In case the two measured entropies have a small difference, the generalisation is not possible and the two kinds of signatures are simultaneously generated. Also in this case, the default rules are set to increase the protection based on the user feedback.

Other ways of generalisation are inherent to the J48 tree construction options. By default, J48 generalises over the class having the largest number of instances in the training data-set. This generalisation is misleading in our case because normal messages may outnumber SPIT messages in the training set.

[2]The pseudo-code is written in Python-like style

Another option is to construct binary trees. In binary mode when we choose an attribute to be a split node the data are split into two classes depending on wether the attribute's value is equal to or different than one chosen value. Also in this mode, the same attribute can be chosen at different levels of the tree.

Two modes of tree pruning are also available: by subtree replacement and by subtree raising. Subtree replacement consists in transforming a node into a leaf while subtree raising consists on moving a node upwards towards the root of the tree. The pruning is controlled through a confidence factor. J48 has an additional pruning method called reduced-error pruning. This method reserves a given proportion of the dataset for testing, builds the tree over the remaining data and tries to minimize the error rate over the testing subset. Finally, J48 allows to put a constraint on the minimum number of instances that can constitute a leaf. Predicting which options would give the best classification accuracy is difficult and dependent on the available dataset.

We note that optimizing the protection policy may need some aggregation functions over the signatures, for instance aggregation of IPs into a subnet, aggregation of geographic locations into one geographic region or aggregation of strings having the same prefix or suffix.

3.2 Protection Policy Update

We assume that the best protection policy at time t comes from a decision tree trained over all the data since $t = 0$. Thus our problem turns into building a decision tree for a very large data-set. Many solutions to this problem in the literature propose to convert the decision trees into rules and the rules are combined into a single rule set [3, 11]. The combination process is based on an approach, suggested in [19], in which rules that match one or more examples but assign them to different classes are resolved. Rules which are similar are combined into more general rules.

Dealing with categorical variables simplifies the combination process since no decomposition is needed. We suggest that the protection policy at time t may be obtained by merging of two other policies:

- the old policy calculated at time $t-T$ representing the previous experience, all the data prior to $t-T$ having been deleted;
- and the new policy calculated over the data-set stored in the interval $[t-T, t]$.

In case of conflict, we consider that recently learnt rules have more weight than old ones (e.g. because of a SPIT signature that became used by benign users or vice versa).

In online mode, we define two FIFO buffers: one for SPIT messages and one for normal messages. At any moment we have two trees: the "final" tree, and the tree under construction. We also define two parameters : w_0 and w_1. w_0 represents the smallest number of SPIT messages required to build a decision tree (e.g. $w_0 = 100$). Each time we have new w_0 SPIT messages, we build a decision tree using all the messages in the SPIT buffer and a number of messages 10 times greater from the normal buffer (e.g. 1000 INVITE messages). Then we combine this tree with the final one. The advantage to set w_0 to a small number is to trigger a fast response against new forms of attacks. w_1 represents

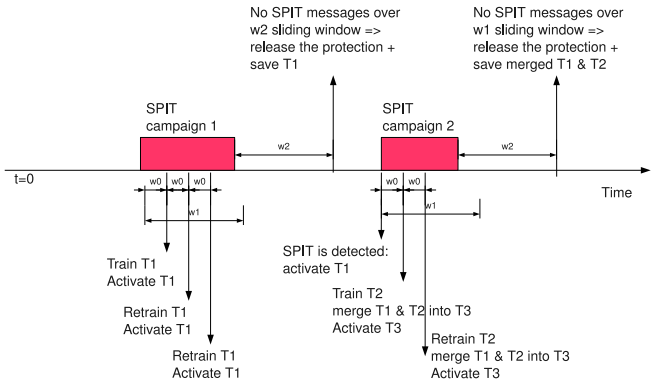

Figure 4: Online Policy Building and Merging

the number of SPIT messages required to consider the learnt tree as final, combine it with the last final one, free the SPIT buffer and start learning a new one (e.g. $w_1 = 1000$ messages). This setting ensures that (1) two different final trees are learnt over two different sets of SPIT messages and (2) our combined knowledge of signatures is directly activated for protection.

Since SPIT messages are usually conducted in form of campaigns, an example of the process in online mode is shown in Figure 4. When we start receiving SPIT messages, we activate the last final tree, start learning a new one and combine the results. When no more SPIT messages are received for a time w_2, we open the system and store the final tree for later use.

3.3 Discussion on the Online Parameters

The choice of the different parameters of the online system is subject to a number of constraints. The first one is the Number of attributes versus w_0 (number minimum of SPIT messages required to build a tree): w_0 must be at least larger than the number of used features, because the J48 algorithm is designed for the cases where the number of instances is larger than the number of attributes. Choosing a large number of attributes builds a better signature but delays the time-to-signature hence the time to apply suitable protection. Minimizing the number of used attributes reports to a problem of feature selection. It is difficult however to estimate which attributes may give a better accuracy. We suggest that the set of attributes may change periodically in order to deflect adversarial learning attacks.

The second constraint is the unbalanced "normal" vs. "spit" sets: The tree is built based on w_0 SPIT messages and 10 times that number of normal messages. Although this setting avoids overfitting small segments of the normal traffic, it produces an unbalanced training set. To alleviate this problem, one can assign weights to the SPIT messages in the dataset. Even so, each time we increase the number of buffered normal messages the number of categories increases as well. That is, using a weighting scheme remains constrained by the size of the memory buffer and by the number of categories that J48 can support. The latter depends on the dedicated Java heap memory size.

Table 1: SIP attributes extracted for experimentation

Attribute	Field	Categories	Entropy
uri_host	Request Line	10	0.36
from_host	From (SIP)	9	0.39
to_host	To (SIP)	9	0.44
o_user	Origin(SDP)	16	0.92
m_codec	Media (SDP)	13	1.27
via_host	Via (SIP)	64	2.3
ip_src	IP Header	82	2.33
user_agent	User Agent (SIP)	13	2.67
contact_host	Contact (SIP)	87	2.74
c_host	Connection (SDP)	86	2.74
o_host	Origin (SDP)	161	6.26
from_user	From (SIP)	400	7.45
contact_user	Contact (SIP)	455	7.65
to_user	To (SIP)	1398	9.88
uri_user	Request Line	1404	9.88
m_port	Media (SDP)	3467	10.96

4. EXPERIMENTATION AND RESULTS

4.1 SIP Parser and Data-Set

Our SIP parser is developed using the Jain-SIP Java library [9]. According to [6], more than 274 attributes per message are available. For our experiments we extract 16 SIP and SDP attributes as shown in Table 1. Fields that are randomized in normal messages such as Call-Ids, tags and branches are excluded. We notice however that these fields may distinguish some attack tools using constant message templates.

Our data set is composed of one SPIT-free trace coming from a real VoIP provider, one trace obtained at a deployed instance of a SIP user-agent honeypot[3] and 3 traces obtained by running 3 SPIT tools in a local test-bed[4].

The normal trace represents 4 hours of calls (5609 INVITE) obtained under a privacy agreement, hence we hide the values of the fields for the normal messages. To better characterize this trace, we show the number of categories found and the corresponding information entropy for each SIP attribute in Table 1 (the attributes are sorted based on their entropies). The SPIT tools we use are:

- Warvox[5] (80 INVITEs)
- Spitter/Asterisk[6] (870 INVITEs)
- SIP Bots[7] (1861 INVITEs)

We adapt the IP of traces by randomly peeking IPs from the normal trace and replace the IPs in the 3 SPIT traces. That is, we assume that the attackers use the same IPs as the legal users. This is the case where IP-based filtering would not be efficient. The trace obtained at the honeypot (24 INVITEs) does not necessarily represent SPIT since no audio was received.

[3] `http://artemisa.sourceforge.net/`
[4] Only INVITE messages towards the SIP firewall are filtered
[5] `http://warvox.org/`
[6] `http://www.hackingvoip.com/sec_tools.html`
[7] `http://voipbot.gforge.inria.fr/`

4.2 SPIT Signatures

We perform a 10-fold training of J48 decision trees using the Weka default parameters (pruning with a confidence factor of 0.25 and a minimum of two instances by leaf). Our extraction tool takes then the learnt decision tree as input and generates the signatures. The results are shown in Table 2. The accuracy is defined as the number of correctly classified instances over the total number of instances.

The results show that even if the disjunction of the rules learnt from each training set is not equivalent to the rules learnt from the overall training set, we have practically the same accuracy results.

4.3 Adversarial Attack Experiments

Our method is expected to be robust against adversarial attacks. Whenever the attacker changes the SPIT INVITE messages to bypass the firewall, the end-users label them as SPIT in nearly real-time (i.e. after the call hang-up). The learning component finds then new signatures and replace the old ones. Assuming that the attacker has not access to the benign traffic and can not handle the end-user feedback, it is hard for him to create messages that can not be distinguished from the legal ones. However, we assume that the attacker may use two "obfuscation" techniques:

Random generation of values: The attacker has identified the feature as important to obfuscate, but in absence of additional information, he blindly generates a large amount of values.

Cloning of values from normal messages: The attacker knows the values of some features in the normal traffic, but he does not know which values appear together in a normal message and the occurrence frequency of each value.

We evaluate 17 different scenarios where the attributes of the SPIT messages are divided accordingly into two sets, C and G. We assign values to the attributes of the set C by peeking randomly one of the corresponding values in the normal trace. For the attributes in set G, the values are assigned by a random string generation function. In each experiment, the training set is composed of the normal trace (5609 messages) and 5600 generated SPIT messages. In the first scenario ($n = 0$), we assume that the attacker knows the values for all the attributes, that is no randomization is used. In the 16 subsequent scenarios ($n = 1, ..., n = 16$), we hide the n attributes from the top of the tree (having the lowest entropies in Table 1). The attacker randomizes the values of these n attributes.

The results are shown in Table 3. we focus on 4 variables: the tree size and depth indicating the tree complexity, the information gain of the root node, the 10-fold J48 accuracy and the 10-fold accuracy of the rule-set formed by the extracted signatures.

The results can be interpreted as follows: in the first scenario, the values of the attributes are the same for the SPIT and the normal messages. However, the occurrences of these values are equally proportional in SPIT and more biased in the normal messages which allows the decision tree to accomplish good accuracy. In the subsequent scenarios, the attributes having the lowest entropies in the normal messages are randomized in the SPIT messages. So, these attributes have high information gains but a large number of

Table 2: SPIT signatures

Training set	10-fold J48 Accuracy	Signature
Normal vs. Warvox	100%	o_user = "root" and m_codec = "PCMU/8000"
Normal vs. Spitter	100%	o_user = "root" and m_codec = "GSM/8000"
Normal vs. SIP bots	99.96%	m_codec = "PCMA/8000" and user_agent = "Twinkle-v1.1"
Normal vs. Honeypot (Real attack IPs)	100%	from_host in ["74.206.180.162", "194.28.112.29", "174.123.134.242"]
Normal vs. Honeypot (adopted IPs)	100%	o_user = "sip" or (o_user = "root" and m_codec = "L16/8000")
Normal vs. All	99.96%	(m_codec ="PCMA/8000" and user_agent ="Twinkle-v1.1") or (m_codec = "PCMU/8000" and o_user = "root") or (m_codec in ["L16/8000", "GSM/8000"])

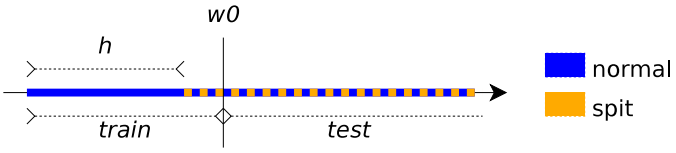

Figure 5: Layout of Messages in an Emergent Attack Scenario

different categories. What happens is that the J48 algorithm decides to avoid overfitting and moves to the next attributes having lower information gains but a smaller number of different categories. We obtain a slightly lower accuracy and complex trees in terms of depth and size. Starting from scenario $n = 5$, even the non-randomized attributes have a large number of different categories (the smallest one is 82 for the IP_src attribute). Therefore, the J48 algorithm stops avoiding overfitting and picks again the attribute with the greatest information gain, that is regardless the number of distinct categories. At the same time, we notice a sudden decrease in the J48 accuracy. In fact, all the random values at the root node that appears in the testing time and were unseen in the training time are considered normal. This is due to the default generalisation of J48 (the normal class has slightly more occurrences in the training set). Our alternative generalisation heuristic classifies correctly the unseen data instances and accomplishes full accuracy. J48 has a similar accuracy if trained and tested with the whole dataset.

4.4 Online Adaptivity Experiments

The goal of these experiments is to evaluate the online behavior of our approach: upon undergoing a new attack, how long does it take to generate a signature and with which accuracy? In other words, how many labeled SPIT messages are needed before converging to a good protection policy? The setting of these experiments is shown in Fig. 5: a history of h normal messages is used as reference, which is followed by a 50/50 mixture of one (new) SPIT attack and normal messages uniformly drawn from our dataset. The tree is built after w_0 SPIT messages have been reported and is trained over all the messages seen since the beginning. the remaining normal and SPIT messages are used for testing.

Experiments with different options for the J48 algorithm (n-ary and binary modes) show that directly activating the learnt decision trees as a protection policy has its limitations. Our approach consists rather on extracting the signatures from the *n-ary* tree by transforming it into a generalised rule-set. Fig. 6 compares our approach to binary and n-ary trees (for $w_0 = 80$). It plots, for each given level of accuracy, the percentage of experiments that successfully reach the required level. The generated rule-set outperforms both binary and *n-ary* trees. The approach shows similar performance under differently unbalanced training sets as controlled by the h parameter. More than 99% accuracy is also obtained using smaller values of $w0$ down to 16 messages and in some cases down to 5.

5. RELATED WORKS

The SPIT problem has retained attention of research and industry in the last few years. Quittek et. al. [14] applied hidden Turing tests on the caller side and compare their results to typical human communication patterns. For passing these tests, significant resource consumptions at the SPIT generating side would be required which contradicts the spammer's objective of placing as many SPIT calls as possible. Nassar et. al. [8] have proposed a set of safeguards that can be activated/deactivated progressively based on a risk model and an anomaly detection system. Quinten et. al. [13] surveyed the anti-SPAM techniques and advised to use complementary approaches. Many systems (e.g. [16, 20]) proposed user feed-back among other modules such as

Table 3: Results for synthetic data

Scenario	Tree size	Tree depth	Information gain (root)	10-fold J48 Accuracy(%)	10-fold Ruleset Accuracy(%)
n=0 (#C=16, #G=0)	58	4	0.66	98.68	98.58
n=1 (#C=15, #G=1)	4645	3	0.62	97.51	98.22
n=2 (#C=14, #G=2)	4665	2	0.61	93.43	98.34
n=3 (#C=13, #G=3)	4786	3	0.61	94.39	97.97
n=4 (#C=12, #G=4)	9451	2	0.53	90.71	97.69
n=5 (#C=11, #G=5)	4624	1	1	65.23	100
n=6 (#C=10, #G=6)	4594	1	1	64.99	100
n=7 (#C=9, #G=7)	4619	1	1	64.78	100
.	.	.	.	.	.
.	.	.	.	.	.
n=16 (#C=0, #G=16)	4188	1	1	66.61	100

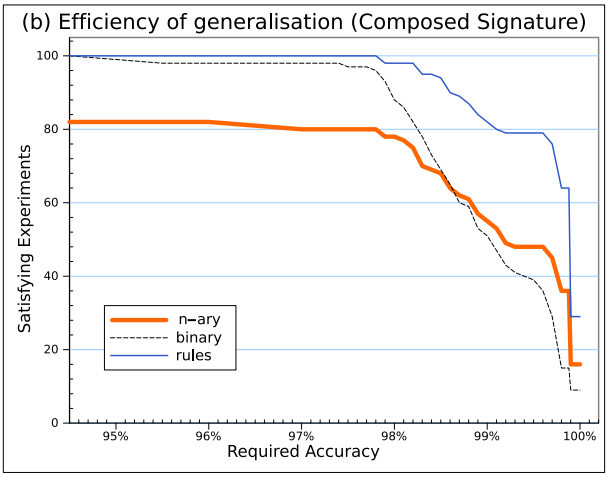

Figure 6: Number of Satisfying Experiments in Function of The Required Accuracy

white/black list, simultaneous calls, call rate and URI's IP/-domain correlation. To our knowledge, no approach really addressed exploiting the user feed-back in order to generate adaptive SPIT signatures from the signaling messages.

Decision trees are a machine learning technique that has been extensively used in many domains including networks and services security. Abbes et. al. [1] addressed protocol analysis for network-based intrusion detection. They defined for each protocol a set of features and used the characteristics of the traffic to build an optimal decision tree. The inputs of their program are a training file (attack, non-attack), a feature set and one criterion for feature selection. Constructing decision trees for sparse categorical attributes as in our case has required attention in [7]. The authors proposed a new algorithm that outperforms other classification algorithms for data-sets where only few attributes are major discriminating attributes. Hess et. al. [6] used decision trees for increasing the rate of acceptance for SIP messages by the SIP parsers. Their approach so-called Babel-SIP is a filter that is put in front of a SIP parser and analyzes incoming SIP messages. When a SIP message is classified as rejected by the decision tree, Babel-SIP suggests an adaptation of the header information which should finally force the acceptance of the message. Their approach can not be applied for SPIT prevention since their decision trees are only based on numerical attributes and the presence/absence of headers.

6. CONCLUSION AND FUTURE WORKS

In this paper, we have addressed the problem of SPIT from a new and unprecedented perspective. Based on the nearly real-time end-user feedback, we have proposed a scheme for generating SPIT signatures in the SIP INVITE messages. Hence it is possible to filter the next SPIT calls before ringing their destinations. The generated SPIT signatures are adaptive to the benign signaling traffic and robust against the adversarial malicious one. The generation of signatures is based on supervised machine learning techniques. We namely investigated decision trees with categorical attributes obtained by parsing the SIP messages. Other techniques inspired from the document classification arena will be studied and compared to our approach in future works. The paper detailed the batch and online modes of our approach. The batch mode consists on training the decision tree over a labeled (spit, normal) data-set and then transforming the tree into an if-else rule-set. In online mode, the successive learnt signatures are aggregated and the possible conflicts are resolved. Experimentation on off-the-shelf SPIT tools showed the efficiency of our approach to find the good signatures. However, experiments show that the J48 decision tree is easily defeated using some obfuscation techniques. Our proposed generalisation approach shows instead good robustness against such attacks. The overall framework provides suitable performance for operational deployment in terms of learning time, required memory, size of the rule-set and the call setup delay. The different parameters of the system (i.e. size of the different buffers and windows) are easily configurable.

Different SPIT signatures may imply different SPIT capabilities. For example, a spitter may break a Captcha test by brute-forcing a DTMF guess. Another spitter may start the call by a human-like congratulation in order to bypass a Turing test. One of the goals of our approach is to provide a framework for applying reinforcement learning techniques hence increasing the efficiency of the filtering process. The reinforcement learning aims at selecting the best rewarded challenge (including drop and accept) when a given signature (spit or normal) is detected. Basically the reinforcement learning maintains a table matching each signature with the best challenge response discovered so far. The table is continuously updated using a trial and error scheme. We will

investigate learning by reinforcement techniques in this context in future works.

Acknowledgment

Mohamed Nassar and Olivier Festor acknowledge the financial support of the EU project Scamstop, FP7–232458. Sylvain Martin (Post-Doctoral Researcher) acknowledges the financial support of the Belgian National Fund of Scientific Research (FNRS). This work is also partially funded by EU project ResumeNet, FP7–224619

7. REFERENCES

[1] T. Abbes, A. Bouhoula, and M. Rusinowitch. Protocol analysis in intrusion detection using decision tree. In *ITCC (1)*, pages 404–408, 2004.

[2] J. Basak. Online adaptive decision trees. *Neural Comput.*, 16:1959–1981, September 2004.

[3] L. Hall, N. Chawla, and K. W. Bowyer. Combining decision trees learned in parallel. In *In Working Notes of the KDD-97 Workshop on Distributed Data Mining*, pages 10–15, 1998.

[4] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11:10–18, November 2009.

[5] M. Handley and V. Jacobson. RFC2327: SDP: Session Description Protocol, 1998.

[6] A. Hess, M. Nussbaumer, H. Hlavacs, and K. A. Hummel. Automatic adaptation and analysis of sip headers using decision trees. In H. Schulzrinne, R. State, and S. Niccolini, editors, *Principles, Systems and Applications of IP Telecommunications. Services and Security for Next Generation Networks*, pages 69–89, Berlin, Heidelberg, 2008. Springer-Verlag.

[7] S.-H. Lo, J. C. Ou, and M.-S. Chen. Inference based classifier: Efficient construction of decision trees for sparse categorical attributes. In *DaWaK*, pages 182–191, 2003.

[8] M. Nassar, O. Dabbebi, R. Badonnel, and O. Festor. Risk Management in VoIP Infrastructures using Support Vector Machines. In *6th International Conference on Network and Services Management - CNSM 2010*, pages 48–55, Niagara Falls Canada, 10 2010.

[9] P. O'Doherty and M. Ranganathan. JAIN SIP Tutorial: Serving the developer community. http://www-x.antd.nist.gov/proj/iptel/tutorial/JAIN-SIP-Tutorialv2.pdf.

[10] M. Preda. Adaptive building of decision trees by reinforcement learning. In *Proceedings of the 7th Conference on 7th WSEAS International Conference on Applied Informatics and Communications - Volume 7*, pages 34–39, Stevens Point, Wisconsin, USA, 2007. World Scientific and Engineering Academy and Society (WSEAS).

[11] F. J. Provost and D. N. Hennessy. Scaling up: Distributed machine learning with cooperation. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 74–79. AAAI Press, 1996.

[12] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[13] V. M. Quinten, R. van de Meent, and A. Pras. Analysis of Techniques for Protection Against Spam over Internet Telephony . In *Proc. of 13th Open European Summer School EUNICE 2007*, July 2007.

[14] J. Quittek, S. Niccolini, S. Tartarelli, M. Stiemerling, M. Brunner, and T. Ewald. Detecting SPIT calls by checking human communication patterns. In *IEEE International Conference on Communications (ICC 2007)*, June 2007.

[15] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. RFC3261: SIP: Session initiation protocol, 2002.

[16] R. Schlegel, S. Niccolini, S. Tartarelli, and M. Brunner. Spam over Internet Telephony (SPIT) Prevention Framework. In *Proc. of the IEEE GLOBECOM Conference 2006, San Francisco, USA*, November 2006.

[17] Y. Soupionis, G. Tountas, and D. Gritzalis. Audio CAPTCHA for SIP-based VoIP. In *Emerging Challenges for Security, Privacy and Trust*, volume 297 of *IFIP Advances in Information and Communication Technology*, pages 25–38, 2009.

[18] J. P. G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Comput. Netw.*, 54:1245–1265, June 2010.

[19] G. Williams. *"Inducing and Combining Multiple Decision Trees"*. PhD thesis, Australian National University, Canberra, Australia, 1990.

[20] Y.-S. Wu, S. Bagchi, N. Singh, and R. Wita. Spam detection in voice-over-ip calls through semi-supervised clustering. In *Dependable Systems and Networks*, pages 307–316, 2009.