

# A multi-task learning model for malware classification with useful file access pattern from API call sequence

Xin Wang and Siu Ming Yiu

Department of Computer Science, The University of Hong Kong, Hong Kong  
{xwang, smyiu}@cs.hku.hk

## Abstract

Based on API call sequences, semantic-aware and machine learning (ML) based malware classifiers can be built for malware detection or classification. Previous works concentrate on crafting and extracting various features from malware binaries, disassembled binaries or API calls via static or dynamic analysis and resorting to ML to build classifiers. However, they tend to involve too much feature engineering and fail to provide interpretability. We solve these two problems with the recent advances in deep learning: 1) RNN-based autoencoders (RNN-AEs) can automatically learn low-dimensional representation of a malware from its raw API call sequence. 2) Multiple decoders can be trained under different supervisions to give more information, other than the class or family label of a malware. Inspired by the works of document classification and automatic sentence summarization, each API call sequence can be regarded as a sentence. In this paper, we make the first attempt to build a multi-task malware learning model based on API call sequences. The model consists of two decoders, one for malware classification and one for *file access pattern* (FAP) generation given the API call sequence of a malware. We base our model on the general seq2seq framework. Experiments show that our model can give competitive classification results as well as insightful FAP information.

## Introduction

Malware continues to be one of the big security threats for both the Internet and computing devices. It can be used for espionage, advertisements promotion, ransom demand and other unauthorised activities on your networks and systems. Due to the ubiquity of malware, automatic tools are usually deployed for malware detection or classification. So many ML algorithms have been applied to the classification problems of malwares.

Classification problems of malwares can be divided into two types: (I) malware detection, which is a binary classification problem and decides whether a sample is benign or malicious; (II) malware classification, which is a multi-class classification problem and outputs the family label of a sample known to be malicious. We refer to them as type I and type II respectively.

There are generally two kind of approaches to analyze malwares that are used to build ML-based malware clas-

sifiers: static analysis and dynamic analysis. Static analysis examines the binary executables directly or after disassembling without really executing them. Diverse static features can be used to build malware classifiers, such as PE header information, n-grams at different granularity levels (Masud, Khan, and Thuraisingham 2008), global descriptors of malware image (Nataraj et al. 2011), section entropy, etc. Usually, compositions of various static features are used to get high accuracy (Ahmadi et al. 2016), which consequently lead to high-dimensional binary sparse feature vectors. Random projections and PCA (Dahl et al. 2013) or feature selection (Lin et al. 2015) are often performed to reduce the dimensions of the input space. However, the problem is that extractions of some static features are already very time consuming and memory intensive. Even worse, various obfuscation techniques (You and Yim 2010) make static analysis difficult nowadays and evasive techniques are available to defeat it (Sikorski and Honig 2012). Dynamic analysis observes the behaviors of a malware usually by actually executing it in a sandbox environment. Malwares are then correlated based on the similarity of their behaviors. Two typical methods of dynamic analysis are control flow analysis and API call analysis. API call information can be extracted from both static analysis and dynamic analysis. Earlier works tend to use a simple frequency representation (Tian et al. 2010) (Shankarapani et al. 2010) of the API calls, whose drawback is evident that API calls in one sequence are treated individually and isolately. The sequentiality of the API calls is such a very important feature that should be considered. Some works extract API call semantics (Zhang et al. 2014) or control flow information (Christodorescu et al. 2005) into graph representations. However, it involves too much manual tweak on graph matching and sophisticated feature engineering. Feature engineering can be hard because it requires specific domain knowledge to design helpful features and involves burdensome deployment and testing. Given the huge amount and ever increasing diversity of the malwares, it is important to build scalable model that can learn features of malware automatically. In this paper, we propose a model that learns representations of malware samples in an unsupervised way.

Furthermore, another big problem of previous works on malware detection or classification is the lack of interpretability. For malware detection, a type I classifier marks

a suspicious sample as benign or malicious without give an understandable reason of the marking. As for malware classification, a type II classifier outputs a family label. However, sometimes just knowing the family label is not very helpful or even meaningless. For example, packing is often used by malwares to hide their real payload, which theoretically can be used by any malwares. Though a classifier can tell you that a malware is packed, it tells nothing about the payload of the packed malware. The payload matters because it is the payload that cause substantial harms to a system and a typical classifier is not able to tell you what the payload does. Due to the flexibility of the payload of a malware, our focus should come to the payload.

Even worse, when it comes to zero-day (Wiki 2016c) malwares, a type II classifier will make a mistake because it can only output predefined and already known family labels. So we want to ask the question of can we provide more interpretability for classification problems of malware? As an answer to this question, we propose to generate an FAP, which is a brief description of file access related behaviors, for each sample rather than just giving the class or family label.

We propose a more interpretable model, multi-task malware learning model, that can be used for malware classification and FAP generation at the same time. Apart from classification, we define an FAP generation task which is combined with classification to provide interpretable information. We construct our model based on the multi-task seq2seq model (Luong et al. 2015). Classification and FAP generation, though quite different, are both defined as seq2seq problems in our model.

In summary, this paper makes the following contributions:

- We propose a novel multi-task malware learning model on raw malware API call sequences. First, low dimensional representations of malware API call sequences are learned by RNN-AE in an unsupervised manner. Then multiple decoders, malware classifier and FAP generator, can be trained under the corresponding supervisions. To the best of our knowledge, this is the first time that multi-task learning is applied to malware learning to provide more interpretability.
- Apart from a malware classifier, we propose a decoder for FAP generation. Inspired by the works of automatic sentence summarization (Rush, Chopra, and Weston 2015)(Nallapati et al. 2016) in Natural Language Processing (NLP), we formulate the FAP generation problem, which can be seen as automatic summarization of API call sequences. As far as we know, this is also the first attempt.
- In our model, we reformulate the malware classification problem with RNN as a special case of seq2seq problem whose output length equals to 1. Instead of obtaining the feature vectors of malware by training an RNN to predict the next API call (Pascanu et al. 2015), we think that learning representations with RNN-AE is more natural and intuitive because RNN-AE has been widely used for representation learning in many other tasks.

## Preliminaries

**Recurrent Neural Networks** Unlike traditional neural network that assume all inputs are independent of each other, a recurrent neural network (RNN) is able to deal with sequential data with variable length and has shown its power in many NLP tasks. The idea is an RNN maintains a hidden state which can be seen as the memory of the network. At each time step  $t$ , the hidden state  $h_t$  is updated by:

$$h_t = f(h_{t-1}, x_t),$$

where  $f$  is an activation function that usually provides non-linearity, and  $x_t$  is the input at time step  $t$ . At each time step, an RNN performs the same calculations on different inputs with the same shared parameters.

An RNN can effectively learn the conditional distribution  $p(\mathbf{Y}|\mathbf{X})$  where output sequence  $\mathbf{Y} = (y_1, \dots, y_{T'})$  and input sequence  $\mathbf{X} = (x_1, \dots, x_T)$ . The length of input  $T$  and that of output  $T'$  can be different. Combined with hidden state update equation above, The conditional probability distribution  $p(\mathbf{Y}|\mathbf{X})$  can be unrolled to:

$$p(y_1, \dots, y_{T'} | x_1, \dots, x_T) = \prod_{t=1}^{T'} p(y_t | h_{t-1}, y_1, \dots, y_{t-1})$$

Each  $p(y_t | h_{t-1}, y_1, \dots, y_{t-1})$  is a softmax distribution over all the input symbols.

**Autoencoders** Autoencoder (AE) is a special neural network which tries to reconstruct its input. An AE consists of an encoder and a decoder, where the encoder map the input to a low-dimensional fixed-length vector from which the decoder recover the input as output. The most important feature of AE is that it learns in an unsupervised way.

## Multi-task Malware Learning Model

The basic seq2seq model can be transformed into multi-task seq2seq learning model (Luong et al. 2015). The multi-task extension can improve the performance of seq2seq model on machine translation centered tasks. According to the number of encoders or decoders, three settings are available in MTL seq2seq model: one-to-many, many-to-one, many-to-many. Our model employs a one-to-many setting, consisting of one encoder for representation learning and two decoders for classification and FAP generation respectively (see Fig. 1).

**Representation learning on API call sequence** API<sup>1</sup> call sequences are usually collected by hooking (Wiki 2016b) while running the samples in a sandbox environment. Compared to other methods, one advantage of classifying malwares using API call sequences is that they are inherently semantic-aware since every single API call is an exact action performed by malware, e.g. creation, read, write, modification and deletion of files or registry keys. One thing we should emphasize is that the semantic information of an API call sequence does not only lies in each single API call, but also lies in the sequence itself.

<sup>1</sup>API in this paper means Windows kernel API by default.

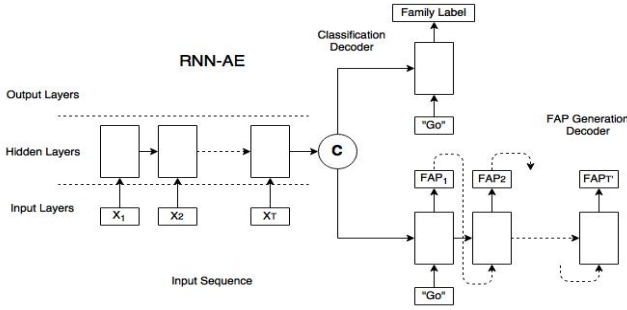


Figure 1: Multi-task Malware Learning Model

RNNs are known for their ability to capture the long term features in sequential or time-series data. An RNN trained to predict next API call can be applied for malware classification (Pascanu et al. 2015). Hidden states of the model are used as feature vectors to train a separate classifier. While in our model, we reformulate the malware classification problem with RNNs as a special sequence to sequence learning (Sutskever, Vinyals, and Le 2014) (also known as seq2seq) problem whose output length equals to one. In a seq2seq model, a sequence is read by an RNN encoder into fixed-length hidden state, which then can be fed to an RNN decoder to predict the output sequence. Both the length of input and output can be variable, making the seq2seq a very general and powerful model for many applications, e.g. machine translation (Sutskever, Vinyals, and Le 2014), text summarization (Nallapati et al. 2016), sentiment analysis (Dai and Le 2015).

RNN-AE maps an input API call sequence to a fixed-length vector  $C$ , which is usually a low-dimensional representation of an input API call sequence.

**Multiple Decoders** The classification task in our model trains a type II classifier and performs malware classification on samples known to be malicious, while the FAP generation task trains an API call sequence summarizer and outputs file access summaries of samples. A decoder shares the same structure as the RNN-AE used for representation learning, and its hidden state is initialized with the accumulated internal state  $C$  (see Fig. 1). A decoder starts with feeding a start symbol “GO”, and generates a distribution. The symbol whose index corresponds to the highest probability in the distribution is the output symbol. Then the symbol generated is fed to the decoder iteratively until the target sequence is fully generated.

Decoders are trained to minimize the cross-entropy over the sequences. Given two discrete probability distribution  $p$  and  $q$ , the cross entropy between them is

$$H(p, q) = - \sum_x p(x) \log q(x) \quad (1)$$

. In the language model, cross entropy measures how accurate the model is in predicting the test data.  $p$  is the true distribution, where the entry corresponding to the true target equals to 1 and others equals to 0.  $q$  is the learned distribution. The closer  $p$  and  $q$  are, the smaller the cross-entropy

Family	Number	Ratio	Mean of Length
Trojan-fakeav	3247	43.7%	228
Adware	2354	31.7%	203
Packed	964	13.0%	320
Worm	865	11.6%	224
Total	7430	100.0%	235

Table 1: Dataset Summarization

is. The loss function is the full cross-entropy over the test dataset

$$L(\theta) = - \frac{1}{N} \sum_{i=1}^N \sum_{j=1}^L \log q_{\theta}(y_{ij}|x_i, C) \quad (2)$$

where  $N$  and  $L$  are the number of the test dataset and the length of the target sequence respectively, and  $\theta$  corresponds to the model parameters.

## Evaluations

We evaluate the accuracy of our model on a public malware API call sequence dataset (Kim 2016) at different granularity levels. We select two datasets which include 7430 samples for coarse-grained evaluation and 4932 samples for fine-grained evaluation (see Tab. 1 and Tab. 6). Malwares whose families are unknown or whose corresponding families do not have enough samples for training are dropped. They are split randomly into train, validation and test datasets, containing 75%, 5%, 20% samples respectively.

## Preprocessing

Supervisions are necessary for the training of decoders. For classification, class labels are already available in the dataset. As to the FAP generation, we first give our definition of FAP and then briefly describe how we extract an FAP for each malware.

**Definition of FAP** : Assume  $S = \{s_1, s_2, \dots, s_n\}$  is a set of file access related APIs, which is called FAP set, and  $l \in \mathbb{Z}$ , then we say  $p = |s|^l$  is an FAP of length  $l$ , where  $s \in S$ .

We select seven file access related APIs from Windows kernel APIs as our FAP set. Different Windows kernel APIs that perform the same function are mapped to one API (see Table 2). For example, both **CopyFile** and **CopyFileEx** perform the function that copy a file, and the difference lies in whether the file already exists. Each function usually have two names for Unicode with **W** as suffix and for ANSI with **A** as suffix respectively. All of them are mapped to **CopyFile** in our FAP set.

We employ a simple way to extract FAPs. We first set the length of our FAP to be  $l_p = |S|$ , where  $|S|$  denotes the number of elements in set  $S$ . Then we generate a binary representation for each malware, which is a binary vector  $v$  with length  $l_p$ . For the  $i$ -th malware,  $v_i[j] = 1$  if the  $j$ -th API in FAP set can be extracted from the  $i$ -th malware API call sequence, otherwise  $v_i[j] = 0$ , where  $j = 1, \dots, l_p$ . We call concatenation of the elements in  $S_i$  the FAP of the  $i$ -th

APIs in FAP set	Original APIs
CreateFile	CreateFileA,CreateFileW
ReadFile	ReadFile
GetTempFileName	GetTempFileNameA,GetTempFileNameW
SetFileAttributes	SetFileAttributesA,SetFileAttributesW
WriteFile	WriteFile
CopyFile	CopyFileA,CopyFileExW
DeleteFile	DeleteFileA,DeleteFileW

Table 2: Mapping of FAP set and original APIs

malware, where  $S_i$  is subset of  $S$  corresponding to the binary vector  $v_i$ . For example, if FAP set  $S = \{a, b, c, d\}$  and  $v_i = [1, 0, 1, 1]$ , then we get  $S_i = \{a, c, d\}$  and  $p_i = "acd"$  after concatenation.

## Model setup

As our model is a multi-task model based on basic seq2seq model, we first experiment on seq2seq model for classification and FAP generation as baselines. Then we verify our multi-task malware learning model on classification and FAP generation tasks respectively. There are many variants RNN units since the advent of LSTM (Hochreiter and Schmidhuber 1997). We use GRU as our default RNN unit, which has been shown to be more computationally efficient than standard LSTM without identifiable loss of performance (Chung et al. 2014). The standard experiment (AE + Decoder) trains on the train dataset and decodes on the test dataset. We can also train on the full dataset (AE(full) + Decoder) because representation learning is unsupervised. We also evaluate on the bidirectional extension of the our GRU-based model with train dataset (bAE + Decoder) and full dataset (bAE(full) + Decoder). For the standard unidirectional RNNs, the output at each time step depends on the inputs at that time step and before. The bidirectional extension allow an RNN has access to both the inputs in the past and future.

## Coarse-grained Evaluations and Results

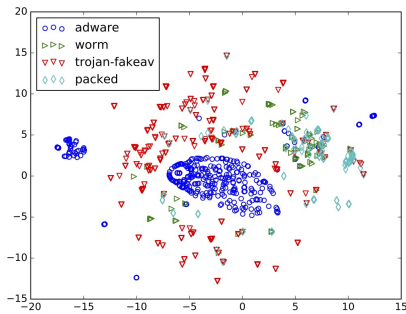


Figure 2: Visualization of Malware Features in Coarse-Grained Evaluation

We experiment on the ground truth. All FAP candidates are mapped to indices (see Tab. 5). We first evaluate on samples from **trojan-fakeav**, **adware**, **worm**, which we denote

Model	$F_3$		$F_3 + \text{packed}$	
	Train	Test	Train	Test
seq2seq classification	99.6%	98.2%	99.6%	96.5%
AE+Decoder	99.6%	<b>97.9%</b>	99.5%	96.5%
AE(full)+Decoder	99.5%	97.7%	99.4%	<b>96.6%</b>
bAE+ bDecoder	99.6%	97.5%	99.6%	96.3%
bAE(full)+ bDecoder	99.5%	97.6%	99.5%	96.3%

Table 3: Performance on classification task

Model	$F_3$		$F_3 + \text{packed}$	
	Train	Test	Train	Test
seq2seq summarization	99.7%	98.7%	99.5%	98.6%
AE+Decoder	98.8%	<b>98.4%</b>	98.8%	<b>98.3%</b>
AE(full)+Decoder	98.7%	<b>98.4%</b>	98.6%	98.2%
bAE+ bDecoder	98.8%	98.3%	98.6%	98.2%
bAE(full)+ bDecoder	98.7%	98.2%	98.5%	98.1%

Table 4: Performance on FAP generation task

with  $F_3$ , and then on samples from  $F_3$  and **packed**. The reason behind this is that we find classification performance on samples from  $F_3$  and packed decreases evidently compared to the performance on samples from  $F_3$ . We visualize the feature vectors in tsne (van der Maaten and Hinton 2008), a method to visualising high-dimensional vectors (see Fig. 2). Adware samples form two clear clusters, while samples from other families are very scattered and some of them are highly interweaved. A feature vector, learned from the API call sequence, can be seen as a *behavioral aggregation* of a malware. In Fig. 2, we can see samples from different families may share very similar behavioral aggregation, while samples from the same family may share quite different behavioral aggregation. That is what a classifier alone can not tell. However, an FAP generator is able to extract the pattern that a malware access the file system regardless of which family the malware belongs to. Unlike classification, the performance of FAP generation does not fluctuate on different datasets. The malwares from different families exhibit obviously different dominant FAPs (see Fig. 3). In our empirical experiments, the trainings with full data or bidirectional extension of GRU do not bring evident increase in performance. As to the connections between some specific FAPs and the malware families, we refer to the following section **Case Studies** for the detailed analyses.

## Case Studies

We analyze the possible correlations between the FAP of a malware and its family in this section. Presumably, malwares from different families may exhibit different FAPs, and those who are in the same family is supposed to have similar FAPs to some extent.

**Worm** A worm is a kind of malware that usually spreads by replicating itself via network. One of the typical actions a worm will perform is "Copy" (sophos 2016a) (sophos 2016b). We can find that the ratio of FAP p6, including "CopyFile", is considerable in the worm samples, yet is ignorable in samples from other family (see Fig. 3(c) and Tab. 5).

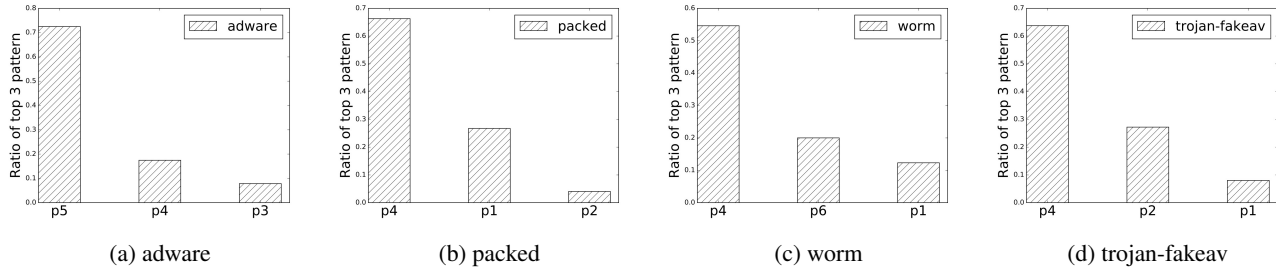


Figure 3: Top 3 FAPs for each family

FAP	ID
CreateFile_WriteFile	p1
CreateFile_ReadFile	p2
CreateFile_WriteFile_ReadFile	p3
CreateFile	p4
CreateFile_ReadFile_GetTempFileName_	
SetFileAttributes_DeleteFile_WriteFile	p5
CreateFile_WriteFile_CopyFile	p6

Table 5: FAP mapping list

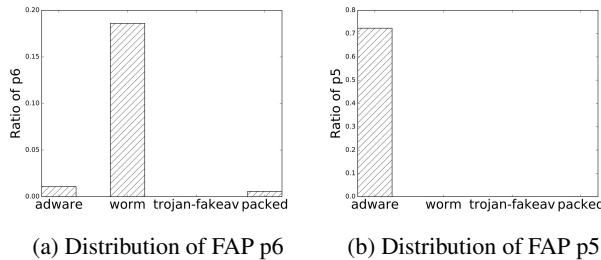


Figure 4: Significance of FAPs in Coarse-grained Evaluation

**Adware** Adware (Wiki 2016a) is a form of malware that downloads and displays unwanted ads when a user is online, redirects search requests to certain advertising websites according to the collected user’s information without the user’s awareness. Generally, an adware will send some HTTP requests according to user’s browsing history. Then store the requested files to **Temp** (www.askvg.com 2016) directory<sup>2</sup> for displaying and drop them in the end (sophos 2016c) (sophos 2016d). We can find that the typical subsequence the dominant FAP for adware (see Fig. 3(a) and Tab. 5) has is **GetTempFileName**, **SetFileAttributes**, **DeleteFile**. **GetTempFileName** create a name for a temporary file and **SetFileAttributes** is followed to set the related attributes of the file. At last, **DeleteFile** is called to drop some files. This typical subsequence can be found in around 70% of the samples of adware and almost none in samples from other families. So the it is highly correlated with behaviours of adware.

<sup>2</sup>**Temp** is a default environment variable in Windows system, which points to a path of folder used to store temporary files. Files in Temp folder are absolutely safe to remove.

**Packed & Trojan-fakeav** Packed malware is a kind of malware that hides the real code of a program through one or more layers of compression or encryption. The top 3 FAPs of **packed** and **trojan-fakeav** are similar and do not include *special* FAPs like **worm** and **adware** (see Fig. 3). Packing or more general obfuscation is usually used as an evasive technique to avoid detection and analysis, which is not exclusive to some specific family of malwares. Because a malware packed does not imply specifically deterministic behaviors, so the API call sequence can be quite similar to that of other samples. The reason of the decrease of classification accuracy on  $F_3$  and **packed** arguably is the intersections of the behaviors of the samples. The interweaved samples from different families are not easy to classify, which necessitate the more interpretable information.

## Fine-grained Evaluations and Results

A dataset with 8 families (see Tab. 6) is selected for fine-grained evaluation. We first evaluate the quality of learned representations by visualising with tsne. The quality of the features is quite self-explanatory, samples from different families form different clusters (see Fig. 5). From the results, we can see both the classification and the FAP generation performance are more competitive than that in coarse-grained evaluation (see Tab. 7).

Compared to other families, feature vectors of some samples from **trojan-fakeav.win32.smartfortress** and **packed.win32.krap** are quite scattered and interweaved (see Fig. 5). This justifies the essentiality of more insightful and interpretable FAP information, because these samples can be very similar and are hard to tell apart from the API call sequences. The fine-grained evaluation also narrow the scope of FAP p6 and p5 to **net-worm.win32.allapple** in **worm** and **adware.win32.megasearch** in **adware**, respectively (see Fig. 6).

## Discussion

**Scalability** It is not enough that ML-based malware model just give us a class label without any explanatory information. On the one hand, given a malware, a classifier can be wrong, and can be much confident or less confident even if it is right. On the other hand, malwares from the same family do not necessarily perform the same way to our system.

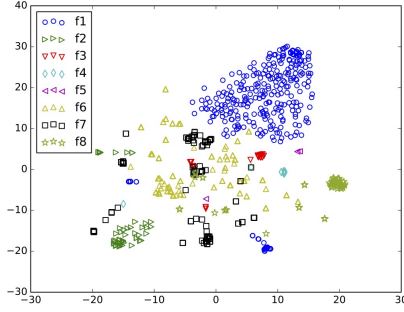


Figure 5: Visualization of Malware Features in Fine-Grained Evaluation

Family	ID	Total	Ratio	Mean*
adware.win32.megasearch	f1	1644	33%	155
adware.win32.downloadware	f2	399	8%	400
adware.win32.screensaver	f3	194	4%	424
worm.win32.wbna	f4	136	3%	470
net-worm.win32.allapple	f5	119	2%	290
trojan-fakeav.win32.smartfortress	f6	1205	24%	217
packed.win32.krap	f7	796	16%	193
downloader.win32.lmn	f8	439	9%	297
Total		4932	100%	231

\* Mean of length of the samples.

Table 6: Dataset Statistics and Family Mapping List in Fine-grained Evaluation

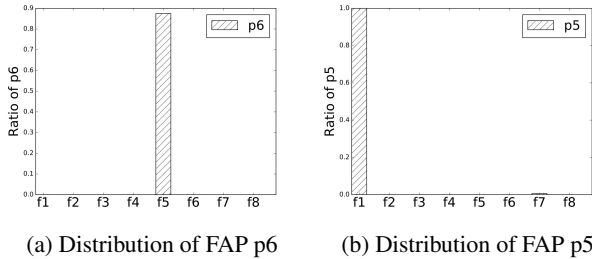


Figure 6: Significance of FAPs in Fine-grained Evaluation

It is natural that we want to know what a malware will do to a system except for the family it belongs to.

Experiment results show that our multi-task malware learning model is able to give FAP as well as class label of a malware. Malware representations learned by RNN-autoencoder from API call sequences are robust enough to

Model	Classification		FAP generation	
	Train	Test	Train	Test
seq2seq summarization	99.9%	99.2%	99.8%	99.3%
AE+Decoder	99.8%	98.9%	99.6%	<b>99.2%</b>
AE(full)+Decoder	99.7%	99.0%	99.7%	<b>99.2%</b>
bAE+ bDecoder	99.8%	<b>99.1%</b>	99.6%	98.9%
bAE(full)+ bDecoder	99.8%	99.0%	99.7%	99.0%

Table 7: Fine-grained Evaluation Performance

trained multiple decoders with quite different objectives and output more informative results.

Rather than performing one single task at a time, our model first learn representations of malwares in an unsupervised way, and then multiple decoders can be trained very efficiently than the training of one single seq2seq task.

**Limitations and Future works** So far, we only use API call sequences for representation learning. Different from classification problems of images and texts, diverse source data can be used for classification problems of malware. Apart from API call sequences, so much additional information, like arguments of API call, structure of the executables, can be leveraged for malware detection or classification. How can we merge these kinds of information into our model to improve the classification accuracy and provide more interpretability is to be done in the future.

Another problem is lack of supervision data, because the decoders are trained in a supervised way. In our evaluation on FAP generation, we adopt a very simple way to craft FAPs from API call sequences and prove its effectiveness. Explorations of decoders with new functions and ways to generate corresponding supervision data are very important to build a robust and informative malware learning model.

## Conclusion

There are two problems of previous works on malware classification: (I) Malwares evolve everyday and new unknown families keep emerging. Classifiers built to output known family labels alone are not enough; (II) Labels themselves are not very interpretable for samples from the same family may perform quite differently even if the label is right. It is more robust to give a brief description to the behaviors of a malware as well as the class label. We build a multi-task malware learning model based on the proven powerful multi-task seq2seq model for classification and FAP generation. An FAP tells what a malware do to a file system, and sometimes it points directly to the family the malware belongs to. Our tentative results show that not only can seq2seq model be used for malware classification based on API call sequences, but can be used for generating more insightful information from the representations learned by RNN-AE. At the same time, unsupervised representation learning enable the model to automatically leverage huge amount of unlabelled data without further feature engineering.

## References

- [Ahmadi et al. 2016] Ahmadi, M.; Ulyanov, D.; Semenov, S.; Trofimov, M.; and Giacinto, G. 2016. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, 183–194. ACM.
- [Christodorescu et al. 2005] Christodorescu, M.; Jha, S.; Seshia, S. A.; Song, D.; and Bryant, R. E. 2005. Semantics-aware malware detection. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*, 32–46. IEEE.

- [Chung et al. 2014] Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.
- [Dahl et al. 2013] Dahl, G. E.; Stokes, J. W.; Deng, L.; and Yu, D. 2013. Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 3422–3426. IEEE.
- [Dai and Le 2015] Dai, A. M., and Le, Q. V. 2015. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, 3079–3087.
- [Hochreiter and Schmidhuber 1997] Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.
- [Kim 2016] Kim, H. K. 2016. Api call sequence dataset. <http://ocslab.hksecurity.net/apimds-dataset>; accessed 11-August-2016.
- [Lin et al. 2015] Lin, C.-T.; Wang, N.-J.; Xiao, H.; and Eckert, C. 2015. Feature selection and extraction for malware classification. *Journal of Information Science and Engineering* 31(3):965–992.
- [Luong et al. 2015] Luong, M.-T.; Le, Q. V.; Sutskever, I.; Vinyals, O.; and Kaiser, L. 2015. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*.
- [Masud, Khan, and Thuraisingham 2008] Masud, M. M.; Khan, L.; and Thuraisingham, B. 2008. A scalable multi-level feature extraction technique to detect malicious executables. *Information Systems Frontiers* 10(1):33–45.
- [Nallapati et al. 2016] Nallapati, R.; Zhou, B.; glar Gulçehre, Ç.; and Xiang, B. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.
- [Nataraj et al. 2011] Nataraj, L.; Karthikeyan, S.; Jacob, G.; and Manjunath, B. 2011. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*, 4. ACM.
- [Pascanu et al. 2015] Pascanu, R.; Stokes, J. W.; Sanossian, H.; Marinescu, M.; and Thomas, A. 2015. Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1916–1920. IEEE.
- [Rush, Chopra, and Weston 2015] Rush, A. M.; Chopra, S.; and Weston, J. 2015. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*.
- [Shankarapani et al. 2010] Shankarapani, M.; Kancherla, K.; Ramammoorthy, S.; Movva, R.; and Mukkamala, S. 2010. Kernel machines for malware classification and similarity analysis. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 1–6. IEEE.
- [Sikorski and Honig 2012] Sikorski, M., and Honig, A. 2012. *Practical malware analysis: the hands-on guide to dissecting malicious software*. no starch press.
- [sophos 2016a] sophos. 2016a. Dabber-c analysis. <https://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/W32Dabber-C/detailed-analysis.aspx>; accessed 16-August-2016.
- [sophos 2016b] sophos. 2016b. Doomjuice-b analysis. <https://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/W32Doomjuice-B/detailed-analysis.aspx>; accessed 16-August-2016.
- [sophos 2016c] sophos. 2016c. Eorezo adware analysis. <https://www.sophos.com/en-us/threat-center/threat-analyses/adware-and-puas/EoRezo> accessed 16-August-2016.
- [sophos 2016d] sophos. 2016d. Installrex analysis. <https://www.sophos.com/en-us/threat-center/threat-analyses/adware-and-puas/InstallRex/detailed-analysis.aspx>; accessed 16-August-2016.
- [Sutskever, Vinyals, and Le 2014] Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.
- [Tian et al. 2010] Tian, R.; Islam, R.; Batten, L.; and Versteeg, S. 2010. Differentiating malware from cleanware using behavioural analysis. In *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, 23–30. IEEE.
- [van der Maaten and Hinton 2008] van der Maaten, L., and Hinton, G. 2008. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research* 2579–2605.
- [Wiki 2016a] Wiki. 2016a. Adware. <https://en.wikipedia.org/wiki/Adware>; accessed 11-August-2016.
- [Wiki 2016b] Wiki. 2016b. Hooking. <https://en.wikipedia.org/wiki/Hooking>; accessed 15-August-2016.
- [Wiki 2016c] Wiki. 2016c. Zero-day. [https://en.wikipedia.org/wiki/Zero-day\\_\(computing\)](https://en.wikipedia.org/wiki/Zero-day_(computing)); accessed 15-August-2016.
- [www.askvg.com 2016] www.askvg.com. 2016. where-does-windows-store-temporary-files-and-how-to-change-temp-folder-location. <http://www.askvg.com/where-does-windows-store-temporary-files-and-how-to-change-temp-folder-location/>; accessed 16-August-2016.
- [You and Yim 2010] You, I., and Yim, K. 2010. Malware obfuscation techniques: A brief survey. In *BWCCA*, 297–300. Citeseer.
- [Zhang et al. 2014] Zhang, M.; Duan, Y.; Yin, H.; and Zhao, Z. 2014. Semantics-aware android malware classification using weighted contextual api dependency graphs. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 1105–1116. ACM.