Method of Learning Malware Behavior Scripts by Sequential Pattern Mining

A.V. Moldavskaya(✉), V.M. Ruvinskaya, and E.L. Berkovich

Department of System Software, Odessa National Polytechnic University,
1, Shevchenko Avenue, Odessa 65044, Ukraine
amme4od@mail.ru, iolnlen@te.net.ua, evg.berkovich@gmail.com

Abstract. Scripts are the knowledge representation model. To our knowledge, there were no machine learning methods for it. In this work we propose a method of discovering script goals and putting them into an order. It is based on sequential pattern mining and regular expressions. This method has been validated by experiments set on malware behavior data. The results show that the discovered goals and their order correspond with expected malware behavior.

Keywords: Scripts · Knowledge representation · Sequential pattern mining · Malware

1 Introduction

Scripts, as the form of knowledge representation, were first proposed during 1970s [1], and yet since that time there wasn't much progress in projecting automated systems based on scripts. The reason for this is in the lack of methods of machine learning for this form of knowledge representation. This leaded to the limited usage of the whole scripts knowledge model. Our aim is to develop such a method. As far as we know, the literature has not discussed the usage of machine learning methods for obtaining a scripts knowledge model.

In Sect. 2, we explore the relevant work which started the theory of scripts. In Sect. 3, we analyze the requirements for a script-based knowledge system. In Sect. 4, we give the description of our method which suggests using sequential pattern mining for learning scripts from a sequence set. As the example field of knowledge, we choose the malicious software and its behavior. The behavior of malicious software often follows the so-called lifecycle, as found, for example, in [13]. The lifecycle describes the typical action sequence performed by a malware of a certain class. This resembles the scripts, therefore we find it appropriate to use scripts for modeling the malware lifecycle. Currently, the knowledge about generalized malware lifecycles (to not be confused with precise behavior knowledge) is only processed manually, as we show in our previous work [2]. Therefore, our choice of knowledge field is, firstly, for explaining and then experimentally testing our method; and secondly, for solving the issue with automated extraction of malware lifecycle models from raw data. In Sects. 4.1, 4.2 and 4.3 we

A. Gammerman et al. (Eds.): COPA 2016, LNAI 9653, pp. 196–207, 2016.
DOI: 10.1007/978-3-319-33395-3_14

provide and explain the results of experiments, showing sequential pattern mining of malware behavior and script learning based on the sequential patterns. Finally, in Sect. 5 we conclude the paper.

2 Related Work

The idea of scripts has two different interpretations: by Schank and by Minsky. Both of them we're comparing in [2]. Basically, Minsky's concept is closer to the frame knowledge model while Schank's is more free-standing. In this paper, we will follow Schank's model. Firstly, we make a short summary of this model, as described in [1–3].

Script is the structured knowledge representation model, used for representing the typical behaviors in some context. The structure of script resembles the one of frames model, except its purpose is to describe the sequences of actions or events. Scripts are used for modeling the human perception, human inference and natural language comprehension. They allow considering the context of the events, which opens the possibility to fill in for the missing parts of incoming information.

Scripts consist of a wide range of component elements, with the most fundamental one being the goal. By achieving goals, the described behavior moves on. The goal can include following elements [1]:

1. The action, which is necessary to complete for achieving the goal
2. The object affected by the action
3. The actor, or the source of performing the action
4. The direction of the action

A basic script is the sequence of goals performed by one actor, representing the single typical behavior in a pre-described context. The generalized goals can be broken into more specific subgoals. A more complicated script can represent multiple behaviors, thus, can include multiple actors. In this case, the notion of role is introduced. The role is the typical behavior of the actor, performed in a certain context. It must be noted that the name of the actor is not important for a script-based analyzing system, and only the role is analyzed.

By further complicating the script, it's possible to group the large, branching goal sequences into scenes. A scene is a group of goals, united by being related to a certain area of the script's context. Scenes describe the typical behaviors for such areas.

Schank proposes enriching the script model even further, by:

1. Adding parameters (modifiers, etc.)
2. Creating categories of scenes and roles
3. Establishing the relations between roles
4. Adding obstacles and distractions

Obstacles and distractions are two wide class of elements found in the incoming information. They are not included in the scripts, because they represent an unexpected external action. Obstacles are the elements which forbid the achievement of the current goal. Distractions are the new unknown goals which interrupt the script following process.

The described features show that scripts open wide possibilities for hierarchic organization of its elements, while still maintaining their order. Following this overview, we compared scripts with alternative existing knowledge representation models, as described in [5]. Frames model is the closest to scripts. The difference is, the objective of frames is to deeply describe the context, while the objective of scripts is to deeply describe the sequences of actions taken within the context. Production rules resemble scripts as well, in the sense that both represent the sequences of some kinds of events. Yet scripts, as it was shown above, feature a number of context-describing parameters, which do not exist in the production rules model. As the result, the inference based on production rules is unable to consider the context – which is exactly the purpose of script-based inference [1]. Semantic networks do not quite resemble scripts, as they do not represent the sequences, just the semantic connections. The common part between scripts and semantic networks is the support for hierarchy.

We've compared the scripts with other knowledge representation models following the comparison methodic described in [6]. As the result, we've marked out the merits and demerits of representing knowledge with scripts. Among the merits, the clearness for human perception and the universality for knowledge fields can be noted. Among the dismerits, the most significant one is the complexity of model learning and the lack of methods to perform it. The main problem of scripts, as Schenk mentions it [3], is the necessity to manually input large amount of data to create the script set. Nevertheless, the manually made scripts are used for building models in the fields related to human perception, linguistics mostly [7,8].

3 The Features of a Script-Based Expert System

The purpose behind a script-based expert system, reasoning system or a script-based decision making system is to expand the incomplete input data by inference, thus discovering new knowledge. For that, the system must solve the following tasks:

1. Accept the set of learning samples, convert it into a set of scripts and store the resulting set
2. Estimate the context of the input and choose the appropriate script, on which the inference will be based
3. Make correlations between the input an the chosen script, while completing the lacking information via inference
4. Take obstacles and distractions into account, in case the area of application demands it

The most important function of such a system is its ability to perform machine learning, meaning that it would build scripts from raw data.

4 The Machine Learning Method for Scripts

Humans are able to interpret the context, because they know the typical regularities and patterns. These regularities are known from observing the environment and the typical events, which occur regularly. This leads to the idea that a script, being the representation of typical events or actions, should be learned from a dataset of events or actions. The machine learning process should be able to derive the generalized, typical sequences. The dataset must have a number of action sequences which happened in the same context in question, or were performed by the role in question. All of these sequences can be different in details, but generally they, being typical, will represent the same situation.

Therefore, to create the method of script machine learning it is necessary to solve two tasks:

1. Discover the regularities, or patterns from the action sequence dataset and convert them into goals and subgoals.
2. Arrange the goals and subgoals in sequence corresponding to the original dataset to obtain a script.

In this paper, the task of finding the regularities in sequences is solved with data mining methods – more specifically, sequential pattern mining methods. These methods did not exist in the years when scripts theory was proposed and expanded, and the computation capabilities of computer systems did not allow processing large data – which is not the question now. Firstly, we make an overview of sequential pattern mining and explore its potential for script learning.

4.1 Sequential Pattern Mining Overview

Sequential pattern mining is the branch of data mining. The object of analysis for sequential pattern mining is a set of sequences. A sequence is an ordered list of itemsets. An itemset is a non-empty set of items [9]. An example of a sequence: S = ⟨{a},{a,b,c}, {b}, {b, c}, {a, d}⟩. If an itemset is composed of items representing the events, then the events within the itemset would be the ones occurring simultaneously. A sequence representing events only coming one after another will be represented with itemsets of one item each.

The goal of sequential pattern mining is to obtain common subsequences from the initial sequence set [10]. These subsequences are named the sequential patterns. The evaluation of how common a subsequence is depends on the chosen method of mining, but generally it is based on the support parameter sup(P), where P stands for the supported pattern. The support indicates how many sequences from the initial set contain a candidate subsequence, i.e. the subsequence which is checked for being a sequential pattern. Once the support

for a subsequence in question hits a certain pre-defined ceiling, the subsequence is considered to be a sequential pattern, and gets placed into the output set.

The main problem of sequential pattern mining, as [10] summarizes, is the output set size. The large amount of patterns makes it difficult to interpret the results. Because of that, it's common to only seek for the patterns with more narrow definitions, which also tightens the requirements for a candidate subsequence to qualify as a sequential pattern. For example, the sequential pattern P_a is closed, if there is no other sequential pattern P_b which would be a supersequence for P_a while $\sup(P_a) = \sup(P_b)$ [11]. Narrowing the definition even further, we can get the maximal sequential pattern, as described in [10,11]. The sequential pattern Pa is maximal, if there is no other sequential pattern P_b which would be the supersequence for P_a . The support is not considered while mining the maximal sequential patterns. So, from a candidate sequences set $\langle P_a , P_b \rangle$, if $\sup(P_a) > \sup(P_b)$, and $P_a \subset P_b$, then P_a and P_b are both closed sequential patterns, but only P_b is a maximal sequential pattern.

4.2 Method of Learning Scripts from Sequential Patterns

The purpose of the following method is to discover and build a set of hierarchic scripts consisting of goals and subgoals in a pre-described context. The method solves both tasks that we've set in Sect. 3. The described method is explained on malware behavior data, although can be extended to handle any kind of behavior data presented in sequences of actions.

The idea behind the method is to apply a two-steps approach. In Step 1, the generalized goals in the form of patterns are discovered by applying sequential pattern mining to the behavior set. The behavior set consists of action sequences. In Step 2, these goals are put in order by referring back to the order of similar actions in the behavior set. The result of these two steps is a set of two-layered hierarchic scripts. The steps can be reiterated for adding new layers to the hierarchy by discovering even more generalized goals.

Let input set D_b^0 of size m be a 0th layer behavior set. In our example, it is, specifically, a malware behavior reports set. Every report is a sequence of WinAPI function names, called by a malware during its execution – i.e. the actions. D_b^0 is analyzed with a chosen algorithm of sequential pattern mining, and the pattern set D_p^1 of size n is discovered as the result. The exact value of n depends on the chosen algorithm and the initial parameters of mining process. We will call $D_p^1 = \{p_1^1 \ldots p_n^1\}$ a 1st layer patterns set, its items we will call 1st layer patterns. The patterns consist of items, specifically WinAPI function names in our case. These items we will call 0th layer items. Both layers are shown on Fig. 1. In terms of scripts, we consider D_p^1 being a set of generalized goals consisting of actions from the 0th layer.

In step two, we propose using regular expressions to solve the task of discovering the order of goals. For the discovered set $D_p^1 = p_1^1 \ldots p_n^1$, a regular expression is created:

$$p_1^1 | p_2^1 | \ldots | p_n^1 \text{ (R1)}$$

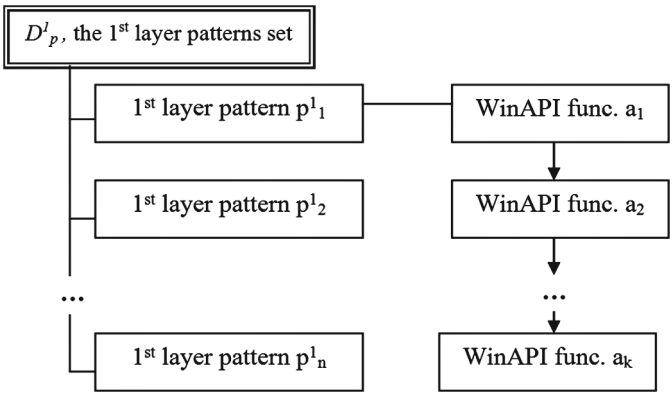

Fig. 1. An example of $1st$ level pattern set, with one of the patterns of length k being expanded on $0th$ layer

R1 is applied to the behavior set D_b^0. This allows discovering the behavior set D_b^1 of size m_1, consisting of ordered sequences of sequential patterns as items. In terms of scripts, D_b^1 consists of scripts, which have a number of goals put in order. Every one of these goals is achieved by performing a sequence of subgoals, each of a size of one action. Therefore, D_b^1 is the output of our method and the set of scripts.

For creating the next layer of the script, D_b^1 is taken as the initial behavior set. After applying a chosen sequential pattern mining algorithm, the set of patterns D_p^2 of size n_2 is received: $D_p^2 = \{p_1^2 \dots p_n^2\}$. D_p^2 consists of sequences made from 1st layer patterns as the items. These sequences, $\{p_1^2 \dots p_{n2}^2\}$, we will call 2nd layer patterns (see Fig. 2).

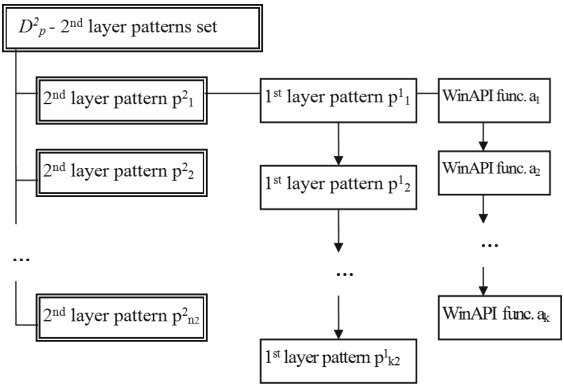

Fig. 2. An example of $2nd$ layer pattern set, with one of the $2nd$ layer patterns of length $k2$ being expanded on $1st$ layer. One of this pattern's items, being a pattern itself, is expanded into $0th$ layer.

It is to be noted that the algorithm for sequential pattern mining on this step is not necessarily the same as on the previous one, when D_p^1 was discovered. The work of an algorithm and its results depend on the length of sequences and the input set size. For D_b^2, as we experimentally discovered, these will be smaller than for D_b^1.

At the next step, the proper order of 2nd layer patterns from D_p^2 is to be discovered. For the discovered set $D_p^2 = \{p_1^2 \ldots p_{n2}^2\}$, a regular expression is created:

$$p_1^2|p_2^2|\ldots|p_n^2 \text{ (R2)}$$

The regular expression R2 is applied to the behavior set D_b^1. The result is the behavior set D_b^2 of size m_2. This set consists of sequences made from 2nd layer patterns as items (see Fig. 3.):

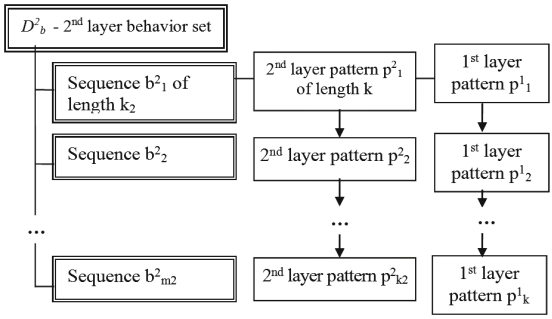

Fig. 3. Behavior set, expanded level by level

To sum it up, the method uses the sequential patterns set D_p^1 discovered from sequences set D_b^0 and from them it discovers the set of sequences of higher level D_b^1, thus making it possible to use the sequential mining. In the terms of scripts, D_b^2 is the set of hierarchic scripts. Every discovered behavior $b_1^2 \ldots b_{m2}^2$ is an hierarchic script. The pre-described context is the malware class from which the behavior reports were taken.

The actor of the discovered scripts, in our case, is the malware which performed the behavior sequences (or a whole malware class, if the set is based on different specimens of a single class at once). The context is the set of parameters for the report-generating system used on the stage of forming the initial behavior set D_b^0. The actor and the context are pre-defined, and not discovered via the mining process, hence we don't mention them across the method description.

4.3 Experimental Evaluation

This subsection demonstrates the application of our method on real raw data on malware behavior. The experiments were done in two phases. Firstly, we discovered the sequential patterns from sequences representing malware behaviors.

For that, we use the existing sequential pattern mining methods. Secondly, we built the scripts based on these discovered patterns. This is done with the usage of our method, as proposed in Sect. 4.2.

The experiment was set with the usage for following data and instruments. The initial behavior datasets corresponded with malware classes. They were formed from the collection of malware behavior reports obtained in [12], which contained 3 157 reports in XML format, presented as WinAPI names sequences. The collection embraces the phases of malware lifecycle which do not require network interaction. The data went through simple preprocessing before applying the sequential pattern mining methods. For creating an input dataset D_b^0 for every malware class, we purged the reports of unknown class malwares as well as the reports of malware families consisting of less than 3 reports. As the result, the input datasets were of following size: Backdoor – 595, Virus – 94, Worm – 224, P2P-Worm – 179, Trojan – 277.

The chosen algorithms of sequential pattern mining were CloSpan and ClaSP, which both discover closed sequential patterns. The difference is the method of data mapping. CloSpan maps the data into a tree, while ClaSP utilizes vertical data representation, which allows higher speed but generates different candidate sequences [11].

We've set the minimal pattern length as 3, meaning that shorter patterns will be purged from the output set. The experiments were held independently from each other, meaning that the discovered patterns were not purged by overlapping. The quantities of 1st layer patterns discovered by CloSpan with according support settings are shown in Table 1.

Table 1. 1st layer patterns, discovered by CloSpan

Class	Total	Patterns discovered, support 100 %–50 %					
		100 %	90 %	80 %	70 %	60 %	50 %
Backdoor	595	0	0	1	49	389	N/A
Virus	94	0	0	0	2	9	12
Worm	224	0	1	8	11	90	998
P2P-Worm	179	0	3	45	253	688	N/A
Trojan	277	0	10	21	38	243	626

The quantities of 1st layer patterns discovered by ClaSP with according support settings are shown in Table 2.

Next, we set an experiment to discover the 2nd layer patterns. For the input, we used the sets of patterns discovered by ClaSP from P2P-Worm and Trojan malware classes, support value 50 %. These were the largest sets obtained at the previous experiment. We applied ClaSP and CloSpan with support 100 %-30 %. The quantities of discovered 2nd layer patterns as the result of this experiment are shown in Tables 3 and 4.

Table 2. 1st layer patterns, discovered by ClaSP

Class	Total	Patterns discovered, support 100 %-50 %					
		100 %	90 %	80 %	70 %	60 %	50 %
Backdoor	595	0	0	23	49	389	N/A
Virus	94	0	3	3	5	14	26
Worm	224	0	0	0	11	90	998
P2P-Worm	179	0	3	45	253	688	1268
Trojan	277	0	10	31	71	318	946

Table 3. The results of 2nd layer pattern mining for P2P-Worm class

Algorithm	2nd layer patterns discovered							
	100 %	90 %	80 %	70 %	60 %	50 %	40 %	30 %
ClaSP	0	0	0	0	0	0	1	2
CloSpan	0	0	0	0	0	1	1	2

Table 4. The results of 2nd layer pattern mining for Trojan class

Algorithm	2nd layer patterns discovered							
	100 %	90 %	80 %	70 %	60 %	50 %	40 %	30 %
ClaSP	0	0	0	0	0	1	1	3
CloSpan	0	0	0	0	0	1	1	4

Consider the specifical example of discovered patterns:

GetProcAddress() – InitializeSecurityDescriptor() – SetSecurityDescriptorDacl() – FreeSid() – GetProcAddress()

This pattern, being a WinAPI names sequence, demonstrates malware's way of working with a process security descriptor.

By further applying our method, as described in Sect. 3, we created scripts that had the discovered patterns as their goals and WinAPI names as subgoals. As an example, one of the scripts for Agent family of Backdoor class appeared as following:

1. GetACP – GetProcAddress – LoadLibraryA This pattern corresponds with the goal of receiving the operational system's code page.
2. GetProcAddress – InitializeAcl – AddAccessAllowedAce – InitializeSecurityDescriptor – RegCreateKeyExA – GetProcAddress – LoadLibraryA This pattern demonstrates how a backdoor achieves the goal of creating the access restriction, by using the security descriptor and putting it at a register key.
3. GetProcAddress – AllocateAndInitializeSid – InitializeAcl – AddAccessAllowedAce – InitializeSecurityDescriptor – RegCreateKeyExA – GetProcAddress – LoadLibraryA This pattern demonstrates how a backdoor achieves the

goal of adding an access restriction, by adding it into an access list, creating a security descriptor and putting it at a new register key.
4. GetProcAddress – AllocateAndInitializeSid – InitializeAcl – AddAccessAllowedAce – InitializeSecurityDescriptor – RegCreateKeyExA – FreeSid – GetProcAddress – LoadLibraryA

This pattern demonstrates achieving the same goal, but the security descriptor is also set free.

Therefore, if we consider the implications behind the discovered goals, we receive the following script (Fig. 4):

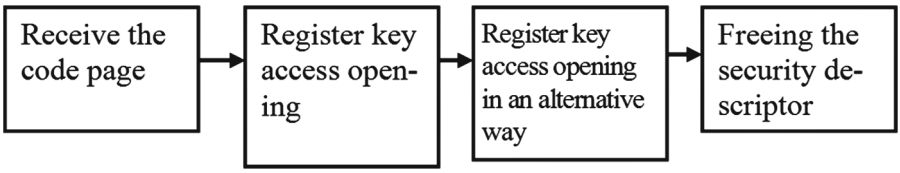

Fig. 4. The script for malwares of Agent family, Backdoor class.

Consider an example of a 2nd layer pattern discovered from Trojan class by ClaSP. It consists of two 1st layer patterns:

1. Pattern p_1^1: RegOpenKeyExW() – LoadLibraryA() – RegOpenKeyExA() – Local-Free() – RegCreateKeyExA() – GetSystemMetrics() – GetModuleFileNameA()
2. Pattern p_2^1 : RegOpenKeyExW() – LoadLibraryA() – RegOpenKeyExA() – Local-Free() – RegCreateKeyExA() – GetModuleFileNameA() – GetVersion()

This 2nd layer pattern demonstrates how Trojan malwares incor-porate themselves into an operational system by consecutively trying to edit the register in two different ways, as shown in Fig. 5:

Another 2nd layer pattern, consisting of 3 1st level patterns, shows the similar behavior.

1. RegOpenKeyExW() – LoadLibraryA() – RegOpenKeyExA() – LocalFree() – Reg-CreateKeyExA() – LoadLibraryA() – RegCloseKey()
2. RegOpenKeyExW() – LoadLibraryA() – RegOpenKeyExA() – LocalFree() – Reg-CreateKeyExA() – RegOpenKeyExW() – LoadLibraryA()
3. RegOpenKeyExW() – LoadLibraryA() – RegOpenKeyExA() – LocalFree() – Reg-CreateKeyExA() – LoadLibraryA() – RegOpenKeyExA()

Therefore, these discovered patterns show different chains of complex actions by which Trojan malwares try to achieve their general goal. The discovered scripts can be used for better understanding of malware behavior and for intellectually malware detection systems.

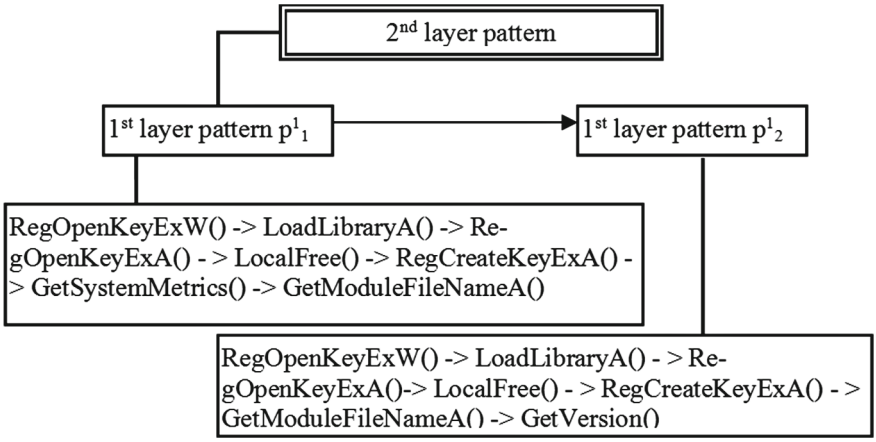

Fig. 5. Multi-layered representation of Trojan behavior

5 Conclusion

In this paper, we discussed the method of applying machine learning to Schank's scripts model. The previous studies of Schank and co-authors [1,3,4] describe in detail the main concepts and elements for a script-based knowledge model, yet they provide no methods of machine learning for it. By reviewing the practical applications of scripts (for example, [7,8]), we discovered that scripts are still being formed manually. Our goal was to create a machine learning method which could've solve the task of forming the most basic and necessary script elements from raw data sequences. For that, we first compared the scripts with other knowledge representation models. Second, we formulated the requirements for a self-learning script-based computer system. Basing on these requirements, we propose the method of script goal mining and ordering. That allowed creating the hierarchic scripts automatically.

The experiments, set on malware behavior dataset, show the following results. 1st layer patterns obtained, for 5 malware classes and support 90 %: 1, 3 10 (3 classes); support 80 %: 1 to 45 (4 classes); support 70 %: 2 to 253 (5 classes); support 60 %: 9 to 68 (5 classes); support 50 %: 12 to 1268. 2nd layer patterns obtained, for 2 malware classes, with support 50 %: 1 each; support 40 % - 1 each, support 30 %: 2 to 4 for each class. With the proposed method of script learning, the discovered patterns were united into scripts. By analyzing the specific examples, we saw that the discovered patterns and resulting scripts make sense and correspond with real malware actions.

The described method was tested only on malware behavior data, however, it can be of interest in analyzing and mining the behavior-related data of any nature. There is a wide range of possible future research based on the suggested method. Our next steps will be towards building a script-based, malware behavior detecting expert system; as well as developing an approach for discovering cyclic script goals among the normal ones. This research is currently in work, and seems to be leading us into developing a new kind of sequential patterns.

References

1. Schank, R.C., Abelson, R.P.: Scripts, plans and goals. In: Proceedings of the 4th International Joint Conference on Artificial Intelligence. IJCAI 1975, vol. 1 (1975)
2. Ruvinskaya, V.M., Moldavskaya A.V., Kholovchuk A.O.: Scenarii kak forma predstavlenia znaniy pro povedenie vredonosnyh program (Scripts as a form of representation of knowledge of malware behavior). Transactions of Kremenchuk Mykhailo Ostrohradskyi National University **4** (2014)
3. Schank, R.C., Riesbeck, C.K.: Inside Computer Understanding: Five Programs Plus Miniatures. Psychology Press, New York (2013)
4. Schank, R.C., Abelson, R.P.: Scripts, plans, and knowledge, pp. 151–157. Yale University (1975)
5. Joseph, Giarratano, Gary, Riley: Expert systems principles and programming, 2nd edn., p. 321. PWS Publishing Company, Boston (1998)
6. Krisilov, V.A., Poberezhnik S. M., Tarasenko R.A.: Sravnitelniy analiz modeley pred-stableniya znaniy v intellektual'nyh sistemah (Comparative analysis of the knowledge re-presentation models in artificial intelligence systems). Odes'kyi Politechnichnyi Universytet. Pratsi, vol. 2, p. 6 (2011)
7. Polatovskaya, O.S.: Freim-scenariy kak tip konceptov (Scene-frame as a concept type). ISTU Bulletin (2013)
8. Kushneruk, S.L.: Teoriya tekstovyh mirov: perspectivy issledovaniya reklamnoy kommunikacii. Political Linguistics **25** (2008)
9. Agrawal, R., Srikant, R.: Mining sequential patterns. In: 1995 Proceedings of the Eleventh International Conference on Data Engineering. IEEE (1995)
10. Gupta, M., Han, J.: Approaches for pattern discovery using sequential data mining. Pattern Discovery Using Sequence Data Mining: Applications and Studies, pp. 137–154 (2012)
11. Mabroukeh, N.R., Ezeife, C.I.: A taxonomy of sequential pattern mining algorithms. ACM Comput. Surv. (CSUR) **43**(1), 3 (2011)
12. Sami, A., et al.: Malware detection based on mining API calls. In: Proceedings of the 2010 ACM Symposium on Applied Computing. ACM (2010)
13. Chen, Z., et al.: Malware characteristics, threats on the internet ecosystem. J. Syst. Softw. **85**(7), 1650–1672 (2011)