Краткие сообщения

Short communications

УДК 004.492.3

СКРЫТАЯ МАРКОВСКАЯ МОДЕЛЬ ДЛЯ ОПРЕДЕЛЕНИЯ ВРЕДОНОСНЫХ УЗЛОВ КОМПЬЮТЕРНОЙ СЕТИ

Я. В. БУБНОВ[1)]***, Н. Н. ИВАНОВ***[1)]

[1)]*Белорусский государственный университет информатики и радиоэлектроники, ул. Петруся Бровки, 6, 220013, г. Минск, Беларусь*

Рассматривается проблема определения вредоносных узлов в компьютерной сети. Активность узлов сети фиксируется с помощью зашумленного детектора с привязкой ко времени. В работе предлагается метод идентификации подобных узлов путем классификации временных рядов активности узлов сети. Метод основан на построении скрытой марковской модели для анализируемого временного ряда и последующем поиске наиболее вероятного конечного состояния модели. Эффективность подхода базируется на предположении, что целевые кибератаки локализованы во времени, а значит, активность вредоносных узлов сети отличается от безопасных.

Ключевые слова: скрытая марковская модель; компьютерная сеть; целевая кибератака; классификация временных рядов.

Образец цитирования:
Бубнов ЯВ, Иванов НН. Скрытая марковская модель для определения вредоносных узлов компьютерной сети. *Журнал Белорусского государственного университета. Математика. Информатика.* 2020;3:73–79 (на англ.).
https://doi.org/10.33581/2520-6508-2020-3-73-79

For citation:
Bubnov YV, Ivanov NN. Hidden Markov model for malicious hosts detection in a computer network. *Journal of the Belarusian State University. Mathematics and Informatics.* 2020;3:73–79.
https://doi.org/10.33581/2520-6508-2020-3-73-79

Авторы:
Яков Васильевич Бубнов – аспирант кафедры электронных вычислительных машин факультета компьютерных систем и сетей.
Николай Николаевич Иванов – кандидат физико-математических наук; доцент кафедры электронных вычислительных машин факультета компьютерных систем и сетей.

Authors:
Yakov V. Bubnov, postgraduate student at the department of electronic computing machines, faculty of computer systems and networks.
girokompass@gmail.com
Nick N. Ivanov, PhD (physics and mathematics); associate professor at the department of electronic computing machines, faculty of computer systems and networks.
ivanovnn@gmail.com

Журнал Белорусского государственного университета. Математика. Информатика. 2020;3:73–79
Journal of the Belarusian State University. Mathematics and Informatics. 2020;3:73–79

HIDDEN MARKOV MODEL FOR MALICIOUS HOSTS DETECTION IN A COMPUTER NETWORK

Y. V. BUBNOV[a], *N. N. IVANOV*[a]

[a]*Belarusian State University of Informatics and Radioelectronics,*
6 Pietrusia Broŭki Street, Minsk 220013, Belarus
Corresponding author: Y. V. Bubnov (girokompass@gmail.com)

The problem of malicious host detection in a computer network is reviewed. Activity of computer network hosts is tracking by a noisy detector. The paper suggests method for detection malicious hosts using activity timeseries classification. The approach is based on hidden Markov chain model that analyses timeseries and consecutive search of the most probable final state of the model. Efficiency of the approach is based on assumption that advanced persisted threats are localised in time, therefore malicious hosts in a computer network can be detected by virtue of activity comparison with reliable safe hosts.

Keywords: hidden Markov model; computer network; advanced persisted threat; timeseries classification.

Introduction

Malicious activity detection methods in corporate computer networks still remain an important problem in computer science. To enforce the network security special instrument are embedded in operating systems and ingenuity software are elaborated. Such applications intend to prevent cyberattacks against the local networks and personal computers, theft of data, undesired spam-advertised products, dangerous drive-by hidden objects infected a visitor's machine with malware. As a result, the growing interest is observed in developing systems to protect the end user from the potential attack.

Except of the information theft, infected hosts are exploit for distributed denial of service attacks through a botnet. In such scheme DNS (domain name system) tunneling is commonly used to control the attack through central server. Hosts usually access the central server through hardcoded IP address or domain name. Therefore, networks protected with domain blocklists can easily prevent communication with remote adversary.

At this moment the most advanced method of botnet control is decentralised interconnection between infected nodes. In this scheme, infected host acts as client and takes role of a central server. Considering that decentralised approach does not exploit a single central server battling with these botnets becomes decently complex problem. There are two most common approaches in the organisation of decentralised botnets exist: (i) fast-flux networks and (ii) domain name generation algorithms.

The fast-flux approach assumes a peer-to-peer communication between the infected nodes of a botnet, where an access to the central server is performed through domain name resolution into multiple IP addresses. That means, multiple nodes in fast-flux networks act as proxies to the original central server. In this scheme even after blocking of the infected hosts, botnet still operates as list of IP addresses constantly rotated.

The domain name generations algorithms target corporate policies where malicious domains are blocklisted. In this scenario, malicious hosts exploit a generator of pseudo-random numbers to guess the current domain name of the central server. The frequent rotation of domain names makes blocklist protection powerless against this approach, since the amount of possible domain name combinations virtually infinite.

Mentioned prerequisites lead to the conclusion that exploit prevention of the corporate network nodes for distributed denial of service attacks require blocking of the infected nodes instead of requesting domain names.

Industry-standard method for dealing with this problem is detector analysing packets passing through the network. Such detector may collect system data from the packets [1] or reads system attributes from the nodes [2]. In monitoring system such as Prometheus, Zabbix, Nagios [3], these findings are stored as time series.

Activity of such systems needs tentative tuning of parameters ranges, that provide successful performance of hard and soft components. An operator may use manual adjustment of the system.

Generally, feasible intervals are set by experiment [4], above all in some systems procedures are prescribed and might be automated. The extreme option against harmful node is cutting out network fragment. The only problem is to find point of time for connection breakage due to time and data loss.

That is, common challenge in information security is to determine the point of time when the manual intervention in network activity is necessary.

This basic approach can separate malicious and infected hosts from the safe nodes. Malware is a standout most thoughtful intimidations for the Internet.

Hidden Markov model for malicious hosts detection

Let the detector estimates probabilities referring transmitted network packet as a malicious one. It is a binary classifier. As a result of a detecting action a probability is assigned to each packet P_t, where t is a point of time. The observable events are a flow of packets P_t, the probability of the packet to be malicious is estimated by the detector as $y_t \in [0, 1]$. Markov chain is defined as the event sequence depicted in fig. 1, each observed event may be at one of the two possible states that correspond to two classes $\tilde{y}_i \in \{0, 1\}$.

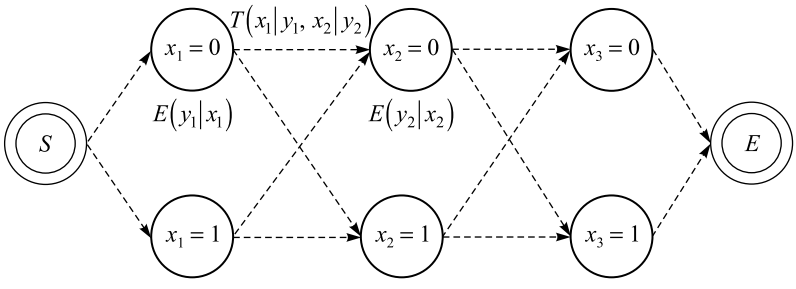

Fig. 1. Markov chain fragment with 3 observed events

It is assumed the i state probability is normally distributed, therefore emission probability of the state can be calculated with the standard formula

$$E\left(y_i|x_i\right) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\left(\frac{x_i - y_i}{\sigma}\right)^2\right\},$$

where standard deviation is chosen as $\sigma = 2.0$ and function is parametrised with a class probability x_i. In other words, the most reliably hidden state is observed in situation when standard deviation of the detector output signal lays inside the boundaries of the standard model.

In most cases observed events form ordinary flow of homogeneous events and mathematically are authentic to Poisson stream. Based on this assumption the transition probability from state i to state j might comply with density of exponential distribution:

$$T\left(y_i|x_i,\, y_j|x_j\right) = \frac{1}{2} \exp\left\{\frac{t_i - t_j}{\exp\left(1 - y_i \oplus y_j\right)}\right\},$$

where $y_i \oplus y_j$ represents the Zhegalkin polynomial, which sets higher probabilities when Markov chain transitions to the states with the same class, whereas class changing is penalised by a factor value e.

For any given ordered time $t \in \mathbf{T}$ interval with N elements the solution of the problem is on the base of reliable events forming the most probable path from initial state t_0 to one of the final state t_N. All intermediate states have to belong to interval $\mathbf{T}$.

Viterbi algorithm can be applied for the most probable way on Markov chain construction (see fig. 1). The algorithm is specified by the following recurrent formulas:

$$V_{1,\,n} = E\left(y_1|x_1\right),$$

$$V_{t,\,n} = \max_i^N \left(E\left(y_t|x_t\right) \cdot T\left(y_i|x_i,\, y_n|x_n\right) \cdot V_{t-1,\,i}\right).$$

Final hidden state is produced by the final solution:

$$x_N = \arg\max_i^N \left(V_{t_N,\,i}\right).$$

Initial problem statement is not getting hidden states of the Markov chain. The challenge is in associating the event series to one of the classes. Direct solving of a classification problem through estimation of observing event sequence in the restored chain produces the result that ignores time locality. In fact, recent events produce more significant effect in comparison with events from the past.

At last, two virtual finite states of Markov chain are defined, they denote that both states are equiprobable. These states are intended for time series classification. Figure 2 represents updated Markov chain (compare with fig. 1).

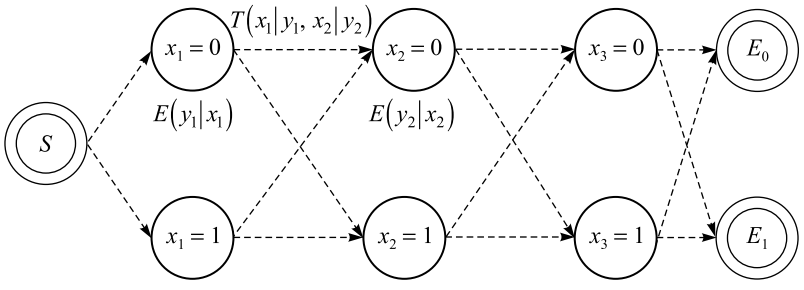

Fig. 2. Extended Markov chain with two extra equiprobable states

Viterbi algorithm estimates two final states probabilities:

$$P(N;\ \tilde{y} = 0) = V_{t_N, N_0},$$
$$P(N;\ \tilde{y} = 1) = V_{t_N, N_1}.$$

Then the result specifying the time series class is calculated by the formula

$$Y = \arg\max\left[P(N;\ \tilde{y} = 0),\ P(N;\ \tilde{y} = 1)\right]. \tag{1}$$

That is, more probable final state defines the chain class.

Dijkstra path probability calculation algorithm

Formula (1) produces the solution of the time series classification problem. Mathematically it is the path in a directed acyclic graph or the most probable path on Markov chain.

Direct solution assumes enumeration of full set of all possible paths from an initial Markov chain's state S to all final states and estimation of the final state probability for each path. It is time consuming approach since due to binary branching at each current chain node an algorithm complexity is exponential, it is exactly equal to power of 2:

$$P(N; \tilde{y}) = O(2^N).$$

Even with progress in processor industry such algorithms are useless. Nevertheless, the optimal path in a given acyclic graph may be found by modified Dijkstra algorithm [5] that has polynomial time complexity.

Modified Dijkstra algorithm applied to Markov chain is described in following paragraphs. Let an oriented graph $G(\mathbf{V}, \mathbf{E})$ presents a Markov chain model. Possible states and edges specify next node transitions. The directed graph under consideration has single initial node and two terminal nodes, that indicate final states with the probabilities pointing to a decision of time series classification. An edge weight here means transition probability $T(v_1|v_2)$ from an edge v_1 to edge v_2. The weight of initial node of Markov chain path is set to $\frac{1}{2}$, an internal node v_2 weight is defined as sum of products of entering node v_1 weight $E(v_1)$ multiplied by transitional edge weight $T(v_1|v_2)$ (see *PathProb*(*) algorithm). The following algorithm specifies weight procedure that calculates final nodes probabilities.

PathProb(G, v_s, v_e)
1) $\boldsymbol{J} \leftarrow \{v_s \rightarrow P_e(v_s)\}$
2) $\boldsymbol{Q} \leftarrow \{v_s\}$
3) **foreach** v **in** $G \setminus v_s$
4) **do** $\boldsymbol{J}[v] = 0$
5) **do while** $\boldsymbol{Q} \neq \varnothing$
6) $u \leftarrow Pop(q)$
7) **foreach** v **in** $G[u]$
8) $p \leftarrow \boldsymbol{J}[u] \cdot T(u|v) \cdot E(u)$
9) $\boldsymbol{J}[v] = \mathrm{Max}(p, \boldsymbol{J}[v])$
10) **if** $v \notin \boldsymbol{Q}$
11) **do** $\boldsymbol{Q} \leftarrow \mathrm{Push}(\boldsymbol{Q})$
12) **return** $\boldsymbol{J}[v_e]$

Description of the main algorithm. The operator 1) initialises a hash table $\boldsymbol{J}$ that stores the graph nodes with corresponding probabilities. Line 2) initialises a nodes queue $\boldsymbol{Q}$ of the nodes to be tested, it is an ordinary FIFO line. Operators 5)–11) set weights to the graph nodes and edges. At each iteration current node is under examination and the following operation proceeds from the current node along the edge with more probable transition. As the result algorithm yields the most probable terminal node.

Actually, because of only forward transitions along nodes of acyclic directed graph the algorithm has polynomial time complexity on nodes number:

$$P(N; \tilde{y}) = O(2N) = O(N).$$

Greedy path probability calculation algorithm

Dijkstra algorithm allows to reach polynomial time complexity of path probability calculation, while the usage of a hash table results in the need to keep all nodes in memory till the final operator. This needs polynomial amount of memory, or $O(N)$.

Considering that state transitions are possible only between two neighbour states, the original graph $G(\mathbf{V}, \mathbf{E})$ can be represented as the following matrix:

$$\mathbf{A} = \begin{bmatrix} (t_1, y_1, x_1') & (t_2, y_2, x_2') & \dots & (t_n, y_n, x_n') \\ (t_1, y_1, x_1'') & (t_2, y_2, x_2'') & \dots & (t_n, y_n, x_n'') \end{bmatrix},$$

where the number of columns is equal to the amount of DNS tunneling detector observations, and each element is a tuple of three elements: t_i – observation timestamp, y_i – observation probability, and $x_i', x_i'' \in \{0, 1\}$ is a probable class of an event.

Matrix construction assumes union of observation timestamps **T** altogether with observation probability **X**:

$$\mathbf{W} = \bigcup_{i \in N} \{t_i, y_i\}.$$

Having a weight matrix **W**, matrix **A** is calculated using a cartesian product of the possible classes set:

$$\mathbf{A}^T = \mathbf{W}^T \times \{x', x''\}.$$

The greedy version of the path probability calculation algorithm is shown below. The algorithm is called greedy as the decision of the next node in a path is taken considering only two current nodes.

MaxProb$(\mathbf{A}, s, i)$

1) $p_1 = \boldsymbol{J} \cdot T\left(\mathbf{A}_{1, i-1} \middle| \mathbf{A}_{1, j}\right) \cdot E\left(\mathbf{A}_{1, i}\right)$

2) $p_2 = \boldsymbol{J} \cdot T\left(\mathbf{A}_{2, i-1} \middle| \mathbf{A}_{2, j}\right) \cdot E\left(\mathbf{A}_{2, i}\right)$

3) **if** $p_1 \geq p_2$

4) **do return** $\left(p_1, \mathbf{A}_{1, i}\right)$

5) **return** $\left(p_2, \mathbf{A}_{2, i}\right)$

GreedyPathProb$(\mathbf{A}, s, e)$

1) $\boldsymbol{J} \leftarrow E(s)$

2) $N \leftarrow |\mathbf{A}_1|$

3) **for** $i = 2$ **to** N

4) $(p, s) \leftarrow MaxProb(\mathbf{A}, s, i)$

5) $\boldsymbol{J} \leftarrow \boldsymbol{J} \cdot p$

6) **return** $\boldsymbol{J} \cdot \underline{T}(s|e) \cdot E(e)$

Greedy algorithm is divided into two functions: *MaxProb* and *GreedyPathProb*. The first function calculates the probability of the transition from the current state s to one of the states from column $\mathbf{A}_i$. Function returns a tuple of the next state and its probability. The operator 1) in *GreedyPathProb* function calculates emission probability of the start state. Operators 3)–5) perform calculation of the cumulative state probability of transition from the start to final state. The operator 4) updates the state s on each iteration based on the joint probability. Finally, operator 6) returns a joint probability of the path $s \to e$.

This described algorithm allows to reduce space complexity of the path probability calculation algorithm from polynomial to constant:

$$P(N; \tilde{y}) = O(1).$$

Timeseries classification results

The algorithm was tested by a problem of detecting DNS tunnel in a computing network. Numerous malicious applications exploit such approach, e. g. programs that steal credit card data from payment terminals [6; 7].

Initial information is a sequence of observed events **Y** ordered by discrete time **T**. The events occur as a result of DNS tunnels registration with binary classifier. Tunnel detector fixes client inquires to domain server. A set of these events are interpreted as Markov chain, the final events are classified and nodes suspected as malicious ones are found out. The principal problem is in detecting infected nodes.

The validation methodology uses a set of safe requests, consisting of the most popular Internet web-sites, and a set of unsafe requests created by the popular tunneling programs, like iodine, tuns, DNScapy, etc. These two sets are constructed based on etalon dataset for DNS tunneling binary classification [8].

Having these two sets, they are organised in series, where each of the series includes a certain percent of unsafe requests. Additionally, it is assumed that attacks are localised in time, therefore unsafe requests are always represented in a sequence. Figure 3 depicts a set of such timeseries, where 20 % of samples relate to unsafe DNS requests. Each request from the timeseries is passed through the detector, described in [9], which calculates the probability of the request being unsafe.

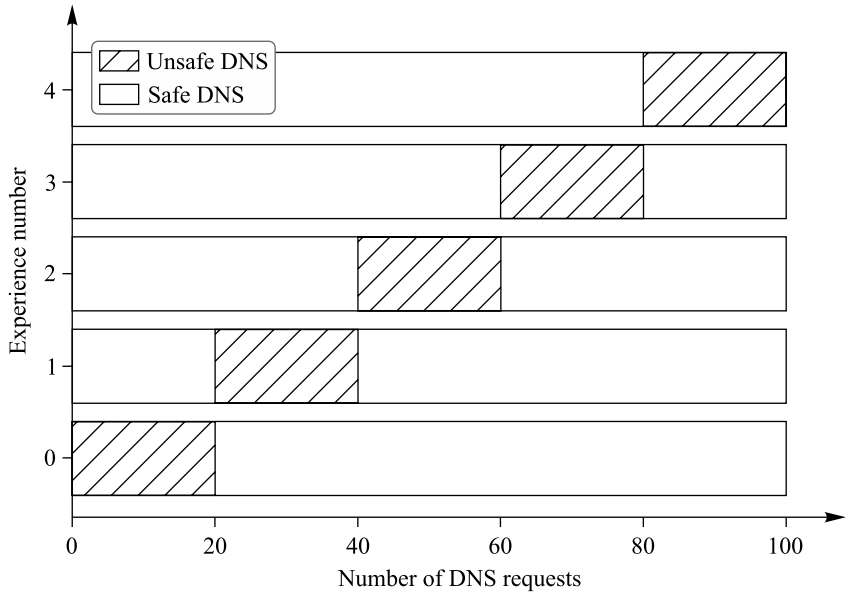

Fig. 3. Ensemble of timeseries used in algorithm evaluation

For each of these ensembles precision and recall are calculated in order to evaluate the proposed algorithm for timeseries classification. Table represents results of classification, where column «ratio» highlights the percentage of unsafe DNS requests in each of the timeseries.

Results of classification estimation

Ratio	Unsafe count	Predicted count	Precision	Recall
0.00	0	0.000	1.000	1.000
0.01	1	0.950	0.860	0.890
0.10	10	9.989	0.987	0.985
0.30	30	30.000	0.995	0.994
0.50	50	50.000	0.994	0.994
0.70	70	69.903	0.999	0.997
0.90	90	90.091	0.999	1.000

The results show high quality of the proposed classifier, which proves the original hypothesis that a sequence of DNS requests represent a Poisson point process, therefore the classification problem can be modeled as hidden Markov model.

Conclusion

The article presents hidden Markov chain as an algorithm for timeseries classification. The algorithm does not need extra infrastructure to store data for computer network analysis, existing system such as widespread Prometheus system may be applied. Besides, algorithm has linear computational complexity and constant space complexity.

References

1. Qi C, Chen X, Xu C, Shi J, Liu P. A bigram based real time DNS tunnel detection approach. *Procedia Computer Science.* 2013; 17:852–860. DOI: 10.1016/j.procs.2013.05.109.

2. Souri A, Hosseini R. A state-of-the-art survey of malware detection approaches using data mining techniques. *Human-Centric Computing and Information Sciences.* 2018;8(1):2–22. DOI: 10.1186/s13673-018-0125-x.

3. Skvortsov P, Hoppe D, Tenschert A, Geinger M. Monitoring in the clouds: comparison of ECO2Clouds and EXCESS monitoring approaches. arXiv:1601.07355 [Preprint]. 2016 [cited 2020 June 2]. Available from: https://arxiv.org/abs/1601.07355.

4. Rong K, Bailis P. ASAP: prioritizing attention via time series smoothing. *Proceedings of the Very Large Data Bases Endowment.* 2017;10(11):1358–1369. DOI: 10.14778/3137628.3137645.

5. Knuth DE. A generalization of Dijkstra's algorithm. *Information Processing Letters.* 1977;6(1):1–5. DOI: 10.1016/0020-0190 (77)90002-3.

6. Deitrich CJ, Rossow C, Freiling FC, Bos H, van Steen M, Pohlmann N. On botnets that use DNS for command and control. In: *7th European Conference on Computer Network Defense; 2011 September 6–7; Gotheburg, Sweden.* Piscataway: IEEE; 2011. p. 9–16. DOI: 10.1109/EC2ND.2011.16.

7. Tatang D, Quinket F, Dolecki N, Holz T. A study of newly observed hostnames and DNS tunneling in the wild. arXiv:1902.08454 [Preprint]. 2019 [cited 2020 June 2]. Available from: https://arxiv.org/abs/1902.08454.

8. Bubnov Y. DNS tunneling queries for binary classification. *Mendeley Data* [Internet]. 2019 [cited 2020 August 17]. Available from: https://data.mendeley.com/datasets/mzn9hvdcxg/1. DOI: 10.17632/mzn9hvdcxg.1.

9. Bubnov Y. DNS tunneling detection using feedforward neural network. *European Journal of Engineering Research and Science.* 2018;3(11):16–19. DOI: 10.24018/ejers.2018.3.11.963.

Received by editorial board 29.06.2020.