







Fingerprinting of URL Logs: Continuous User Authentication from Behavioural Patterns

Jakub Nowak¹ , Taras Holotyak², Marcin Korytkowski¹ ,
Rafał Scherer¹ , and Slava Voloshynovskiy² 

¹ Częstochowa University of Technology, Al. Armii Krajowej 36,
42-200 Częstochowa, Poland

{jakub.nowak,marcin.korytkowski,rafal.scherer}@pcz.pl

² Department of Computer Science, University of Geneva, Geneva, Switzerland
svolos@unige.ch

Abstract. Security of computer systems is now a critical and evolving issue. Current trends try to use behavioural biometrics for continuous authorization. Our work is intended to strengthen network user authentication by a software interaction analysis. In our research, we use HTTP request (URLs) logs that network administrators collect. We use a set of full-convolutional autoencoders and one authentication (one-class) convolutional neural network. The proposed method copes with extensive data from many users and allows to add new users in the future. Moreover, the system works in a real-time manner, and the proposed deep learning framework can use other user behaviour- and software interaction-related features.

Keywords: URL logs · Computer networks · Continuous authentication · Behavioural biometrics · Software interaction · Autoencoder · Convolutional

1 Introduction

For the past twenty years, the Internet and its utilisation have grown at an explosive rate. Moreover, for several years computer network users have been using various devices, not only personal computers. We also have to manage with many appliances being constantly online and small Internet of Things devices. Efficient computer network intrusion detection and user profiling are substantial for providing computer system security. Along with the proliferation of online devices, we witness more sophisticated security threats. It is possible to enumerate many ways to harm networks, starting from password weakness. Malicious software can be illicitly installed on devices inside the network to cause harm, steal information or to perform large tasks. Another source of weakness can be Bring Your Own Device schemes, where such devices can be infected outside

the infrastructure. At last, social engineering can be used to acquire access to corporate resources and data.

Each network user leaves traces, some of them are generated directly by the user, e.g. on social networks, others are closely related to the computer network mechanisms. Thanks to network traffic-filtering devices, network administrators nowadays have an enormous amount of data related to network traffic at their disposal. Authorising users based on their behaviour can be done in many ways, depending on the available data and methods. Identification can be based on facial features [15, 17] or based on spoken instructions [2]. In [3, 11] data from smartphone sensors were used to analyse user behaviour. Similarly, the way how the user unlocks the smartphone [4] can be explored, and based on the collected data, they show the uniqueness of using the phone. This is related to certain user preferences, habits as well as to the physical conditions of individual users, i.e. the way the phone is held. Another option is to authenticate the user with the signature [16]; in this solution, a signature is not only analysed as an image but also the dynamics of the signature creation using a haptic sensor.

In our solution, we test whether the logged-in user has access to a given resource and does not impersonate someone else by breaking initial security measures based on, e.g., a password. Our research can be used in new generation firewall devices working in layer 7 of the OSI model however, in our case, the security rules will be based on the analysis of the pages visited. Our method provides a continuous authentication based on software interaction patterns.

Last years brought learned semantic hashes to information retrieval. Semantic hashing [13] aims at generating compact vectors which values reflect semantic content of the objects. Thus, to retrieve similar objects, we can search for similar hashes which is much faster and takes much less memory than operating directly on the objects. The term was coined in [13]. They used for the first time a multilayer neural network to generate hashes and showed that semantic hashing obtains much better results than Latent Semantic Analysis (LSA) [7] used earlier. A similar method for generating hashes can be using the HashGAN network [5]; this solution is based on generative adversarial networks [8].

In the presented solution, we use autoencoders to create semantic compact hashes for the behaviour of computer network users from their URL request sequences. Our approach is sparked by the aforementioned studies that use hashes to analyze data, especially in NLP. After training the autoencoders, we use the encoder parts to generate hashes and fed them to the input of a one-class convolutional network that performs the final user authentication (Architecture 2). Schematic diagram of the system located in the computer network infrastructure is presented in Fig. 1. We also propose two smaller systems (Architecture 1 and 3) with worse accuracy. Through this research, we highlight the following features and contributions of the proposed system.

- We present three different approaches to URL-based computer network user software interaction behavioural continuous authentication. Up to now, the network traffic was usually analysed by some hand-crafted statistics.

- Our work provides new insights, showing that the system of autoencoders and convolutional neural network can be trained efficiently for one-class authentication for nearly any number of users.
- The method can use nearly any kind of data as a features.
- The proposed system is fast and can be used in real-time in various IT scenarios.

The remainder of the paper is organised as follows. In Sect. 2, we discuss the problem of behavioural authenticating users in computer networks. The proposed data representation and three Architectures are presented in Sect. 3. Experiments on real-world data from a large local municipal network, showing accuracy and a comparison of three presented Architectures, are shown in Sect. 4. Finally, conclusions and discussions of the paper are presented in Sect. 5.

2 Problem Formulation

The aim is to create an additional security layer to verify users in IT systems using data collected by computer network administrative infrastructure. The additional authorisation is carried out without the user's knowledge. The proposed system constantly monitors HTTP request patterns for every computer in the network. The requests come from browsing websites or applications sending queries with URL addresses. In other words, the method provides a software interaction-based behavioural biometrics. It should be remembered here that the addresses stored in the firewall logs apply to both pages opened in WWW browsers and programs running in the background, such as anti-virus or operating system updates. This article is based on data collected from a LAN network infrastructure, which is used by residents of four districts in Poland, as well as network users who are employees of the local government offices and their organisational units, e.g. schools, hospitals, etc. Internet access to the analysed network is done with the help of two CISCO ASR edge routers that route packets using RIP version 2. The data for neural network training was collected between June 2017 and January 2018, and for testing in February 2018. The size of raw logs was approximately 460 GB, and 0.9 GB after preprocessing (selecting time, date, user ID and URL).

Based on the previously collected data, we examine whether an anomaly occurred, which is supposed to indicate a possible attack or use of the account by another user. To solve the problem, we use autoencoders and convolutional networks using various methods of data representation. We divided the task into several stages. The first is to create a session based on registered URL logs from the database. In our case, a session means a set of consecutive, non-repeating URLs for a given user. Each URL address was truncated to 45 characters. This size comes from the average length of addresses and observation of their construction. We have assumed that the most important information is at the beginning of the address. We primarily care about the domain of the website visited, the protocol that was used, and the basic parameters of the GET method. When creating a session, it happened that a query consisting of several URLs was sent

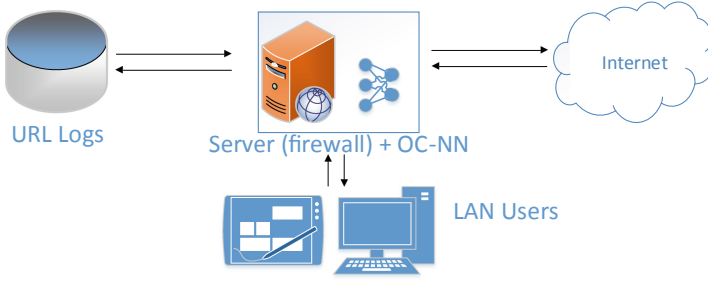


Fig. 1. System location in the computer network infrastructure.

at the same time. In this case, certain sets of addresses were associated in a very specific way; between two identical addresses, another one different from the others was added. To remove duplicate addresses, an additional sorting by address name was used. An example of a set requested at the same time is “address1 address2 address1 address3 address1”. We tend rather to have the set “address1 address2 address3”. This is an exceptional case; however, omitting increases the authenticating CNN error. The session to be analyzed consists of a minimum of 20 different URLs with a maximum length of 45 characters included in the dictionary. Another limitation was the maximum size of the session to be analyzed. We used up to 200 different URLs for fast neural network operation. The interval between the recorded addresses in one session cannot be longer than 30 min. If this time has been exceeded, successive addresses form a new session.

3 Neural Network Architecture

We scrutinized three neural configurations: Architecture 1 with a convolutional network (CNN) with two-dimensional filters used for text classification, Architecture 2 consisting of one-class CNN with unique autoencoder for every user, and Architecture 3 with one-class CNN network and one autoencoder for all the users.

3.1 Data Representation

A URL is an address that allows locating a website on the Internet. The user encounters it mainly when using a web browser. However, URLs can be requested by applications running in the background such as antiviruses, system updates, etc. Each user’s computer uses different applications and at a different time, which allows to even better distinguish them. Very often text is represented by some dictionaries. The construction of the URL has been repeatedly addressed in various articles [1, 20]. The majority of the previous works created some hand-crafted features based on URL statistics. URLs do not consist regular words; thus using word dictionaries is not viable. In our experiment, we encode the entire

address without dividing its subsequent parts. The condition is that the characters that constitute the URL should be in the previously defined set containing 64 unique characters. We were inspired here by Zhang et al. [19] to present text data in the form of a one-hot vector at the character level. The dictionary consisted of the following characters:

```
abcdefghijklmnopqrstuvwxyz_0123456789
-;.:!?:/\| @#$$%^& * ~'+=<>() []
```

If a character was not from the above alphabet it was removed. At the input of the neural network, in addition to the session of addresses used for classification, we also provide user identification data. In our case, we concatenate the user ID data with the URL input session. Therefore, in one input column, we have two values equal to 1, the rest of the rows of one column are filled with zeros. The first one is placed on the position in the range $\langle 1, 64 \rangle$ which defines the letter from the dictionary, the second one on the position in the range $\langle 65, 119 \rangle$ denoting the user ID, because each user has its unique number. The input size to the CNN network in Architecture 1 was 119×4096 , where 4096 means the maximum length of the session we can provide at the network input. The coding scheme is presented in Fig. 2.

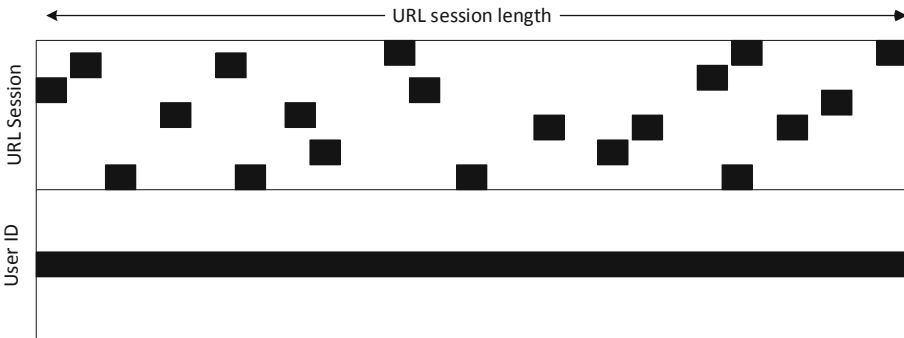


Fig. 2. URL text coding scheme for convolutional networks in Architectures 1–3. The upper part is one-hot character-level text encoding, and the lower part is one-hot user ID encoding.

3.2 Architecture 1

Our first attempt was to use a convolutional neural network with two-dimensional filtering presented in Fig. 3, similar to [10]. The network architecture is as follows:

```

ConvLayer 128 FMs, filter 7x119, stride 3, ReLU
MaxPoolingLayer 3
ConvLayer 256 FMs, filter 5x128, stride 2, ReLU
MaxPoolingLayer 3
ConvLayer 256 FMs, filter 3x256, stride 1, ReLU
MaxPoolingLayer 3
Fully connected 512 + Dropout, ReLU
Fully connected 256 + Dropout, ReLU
Output 2 softmax

```

This method proved to be a weak solution to the problem because the single convolutional neural network had to cope with a highly complex problem and demonstrated a significant classification error. The accuracy of the anomaly recognition exceeded barely 63%, which is slightly higher than the random response and not viable in real-world computer network infrastructure.

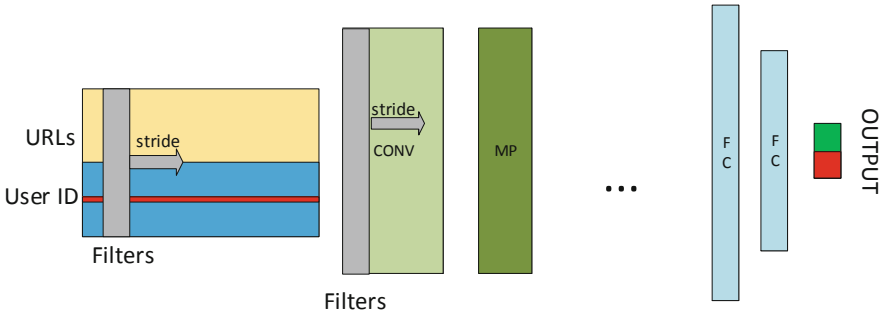


Fig. 3. Convolutional network used in Architecture 1. Detailed meta-parameters are provided in the text. All the filters are large enough to use a one-directional stride. CNNs in Architecture 2 and 3 are similar with different meta-parameters, and the autoencoder latent space instead of the URL session.

3.3 Architecture 2

Architecture 2 was inspired by the one-class neural network [6], and here each user (class) has a different autoencoder. The task of the network is to detect an anomaly in a given class. In our case, we add the user ID to “ask” the network whether given network traffic belongs to this particular user. Initially, we tried to use modified convolutional networks of the U-Net [12] structure without connections between feature maps of the same size (skip connections), what transpired to have an unacceptable training error. In the decoder part of the autoencoder, we implemented a sub-pixel convolutional layer in one dimension inspired by [14]. It changes the size of the input to the convolutional layer by increasing the width of the channels at the expense of their number.

Training data for the autoencoder is created similarly to one sentence in NLP consisting of URLs instead of words. We do not use a separator between URLs; addresses are given as words in a sentence. The structure of the autoencoder for text is different from the structure of the autoencoder for images; here we were inspired somehow by [18]. In our case, the latent space (bottleneck) layer in the autoencoder is 128×64 . In the adopted architecture with pooling, each addition of a layer reduces the size of the smallest, latent space layer in the autoencoder. The size of the latent space is a trade-off determined experimentally between the accuracy and the input size to the one-class CNN. The autoencoder structure used in the article is presented in Fig. 4a), and the detailed meta-parameters are as follows:

- Encoding part
 1. ConvLayer1 64 FMs, filters 3×64 , stride 1, ReLU + padding + MaxPooling
Output 64×2048
 2. ConvLayer1 64 FMs, filters 3×64 , stride 1, ReLU + padding + MaxPooling
Output 64×1024
 3. ConvLayer1 64 FMs, filters 3×64 , stride 1, ReLU + padding + MaxPooling
Output 64×512
 4. ConvLayer1 64 FMs, filters 3×64 , stride 1, ReLU + padding + MaxPooling
Output 64×256
 5. ConvLayer1 64 FMs, filters 3×64 , stride 1, ReLU + padding + MaxPooling
Output 64×128 (after training, it is the input for the discriminator)
- Decoding part
 1. ConvLayer1 64 FMs, filters 3×32 , stride 1, ReLU + padding + UpPooling
Output 64×512
 2. ConvLayer1 64 FMs, filters 3×32 , stride 1, ReLU + padding + UpPooling
Output 64×512
 3. ConvLayer1 64 FMs, filters 3×32 , stride 1, ReLU + padding + UpPooling
Output 64×1024
 4. ConvLayer1 64 FMs, filters 3×32 , stride 1, ReLU + padding + UpPooling
Output 64×2048
 5. ConvLayer1 64 FMs, filters 3×32 , stride 1, ReLU + padding + UpPooling
 6. ConvLayer1 64 FMs, filters 3×32 , stride 1, Sigmoid activation
Output 64×4096

Our solution also utilizes a discriminator as a convolution network. The problem posed by us was to check whether the recorded session belongs to the user and whether a given set of URLs could be generated by a specific user. It was, therefore, necessary to create a suitable discriminator for the autoencoder. The idea of the discriminator is similar to the GAN network [8]; we only care about assessing the mapping of data in the autoencoder and whether the given set could be created by a specific user.

The user identifier has been moved in relation to Architecture 1 from the system input to the discriminator input, i.e. the last coding (latent space) layer in the autoencoder, the user coding method is identical as in the case of Architecture 1. The autoencoder, in this case, is unique for each user (unlike in Architecture 3). The discriminator input in Architecture 2 and 3 was 183×64 , where 183 is made up from 128 (column size with hash from the autoencoder latent

space) and 55 (user ID added in the same way in Architecture 1). The value 64 comes from the number of feature maps from the autoencoder. Detailed meta-parameters of the discriminator in Architecture 2 and 3 are as follows:

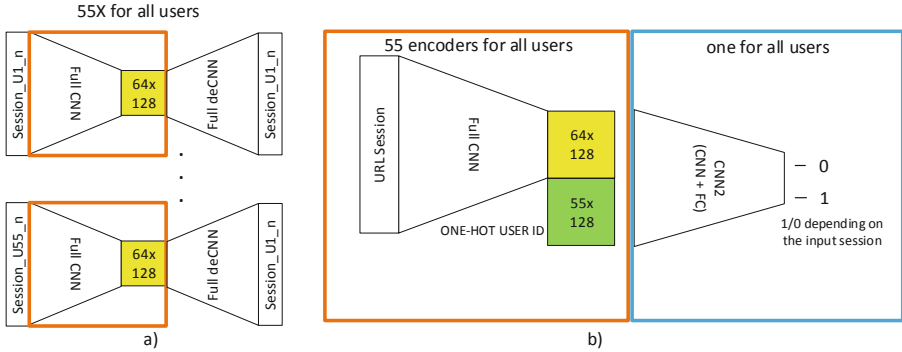


Fig. 4. Architecture 2 consists of many autoencoders and one convolutional network (discriminator). The left part a) is a set of as many autoencoders as the users in the network. The right part b) is the authenticating infrastructure using encoders from a) and the convolutional network with fully connected authentication layer.

- Input from autoencoder + User ID 183×64
- ConvLayer 32 FM, filters 5×183 , stride 2, batch normalization, ReLU
- MaxPoolingLayer 2
- ConvLayer 64 FM, filters 3×32 , stride 1, batch normalization, ReLU
- MaxPoolingLayer 2
- ConvLayer 128 FM, filters 3×64 , stride 1, batch normalization, ReLU
- MaxPoolingLayer 2
- Fully connected 512 + dropout 0.5, ReLU
- Fully connected 128 + dropout 0.5, ReLU
- Output 2 softmax

3.4 Architecture 3

We combined the two previous frameworks to create something in between in terms of size and complexity. Creating separate autoencoders for each user is somehow problematic in terms of logistics, where it is easy to make a mistake in processing sessions for the user. Here we use one autoencoder as a uniform way of representing URLs for all users. The size of the autoencoder is the same as in the previous Architecture 2. This solution improved the results of the CNN from Architecture 1; however, it was worse than Architecture 2 with user-wise unique autoencoders.

4 Experiments

We performed experiments on a database with logs of visited URLs for 55 users. In the case of our database, the most active user had 6,137 training sessions, and the least active user had 440 URL sessions, which was about 14 times less (details are presented in Table 3). In training all Architectures 1–3, we had to generate illegitimate user sessions for the training purposes. We did not decide to create synthetic user sessions because this is a challenging issue, and it could result in generating data different from the existing distribution. In our research, the data that the discriminator has to evaluate negatively was created based on existing sessions. However, we provide them to the network as if they belonged to another user. These sessions are randomly selected from among the entire dataset.

The cross-entropy with softmax loss function from the CNTK package was used to train CNNs (discriminators). The training coefficient for CNNs was taken on as follows: 0.0008 for 5 epochs, 0.0002 for 10, 0.0001 for 10, 0.00005 for 10 epochs, and 0.00001 from then on to a maximum of 300 learning epochs.

We used the binary cross-entropy loss function for training all the autoencoders. The best universal effects for each user were obtained when the learning coefficient was 0.0001 for the first two epochs then 0.00001 for 200 epochs. We trained all the architectures with the momentum stochastic gradient descent (SGD) algorithm with momentum set at 0.9 for both autoencoder and CNN.

To assess the accuracy of the autoencoder (Table 2), the Sørensen similarity coefficient $QS = \frac{2C}{A+B}$ was used, where A and B are compared elements, in our case the output and input to the autoencoder and C is the number of elements common for both layers. After each convolutional layer we used Batch Normalization [9] with RELU activation function. The results are summarized in Table 1.

Table 1. Overall accuracy of the three presented Architectures

Architecture	Accuracy
1. CNN	61.00 %
2. OC and 1 autoencoder	68.00 %
3. OC and 55 autoencoders	84.24 %

The best solution turned out to be Architecture 2 using dedicated autoencoders and one discriminator. The detailed results for each user are presented in Table 3. The number of training sessions for a given user does not affect accuracy.

During the implementation of neural networks, the only limitation turned out to be the available GPU memory. We used four Nvidia GTX 1080 Ti graphics cards with 11 GB of memory. In the case of training autoencoders, we could divide each task into four graphics cards without the need for special techniques enabling multiprocessing on multiple graphics cards. Autoencoder networks can

Table 2. Accuracy of the encoders trained on user URL sessions. After training, the encoder parts are fed to the one-class CNN (discriminator).

User ID	1	2	3	4	5	6	7	8	9	10
Accuracy	87.4%	83.4%	87.5%	83.3%	83.5%	89.8%	85.9%	79.0%	87.3%	87.3%
User ID	11	12	13	14	15	16	17	18	19	20
Accuracy	87.1%	88.1%	89.0%	77.5%	85.0%	87.0%	86.2%	83.5%	85.1%	86.7%
User ID	21	22	23	24	25	26	27	28	29	30
Accuracy	83.4%	87.5%	85.9%	83.2%	75.9%	82.9%	82.5%	84.1%	86.3%	80.5%
User ID	31	32	33	34	35	36	37	38	39	40
Accuracy	77.1%	79.8%	80.4%	77.9%	80.9%	90.8%	72.8%	87.4%	79.9%	88.9%
User ID	41	42	43	44	45	46	47	48	49	50
Accuracy	86.2%	84.7%	78.4%	86.8%	81.9%	87.3%	91.6%	77.8%	79.2%	80.5%
User ID	51	52	53	54	55					
Accuracy	73.9%	74.9%	82.0%	88.5%	85.4%					

Table 3. Number of training and testing sessions, and accuracy of detecting anomalies by the global, one-class CNN for every computer network user.

User ID	1	2	3	4	5	6	7	8	9	10
Training	6137	5199	3184	2892	2434	2368	2120	1928	1841	1674
Testing	1534	1299	795	723	608	591	530	481	460	418
Accuracy [%]	93.82	85.84	82.92	70.59	80.13	93.06	78.05	84.98	83.93	89.24
User ID	11	12	13	14	15	16	17	18	19	20
Training	1663	1650	1266	1236	1206	1181	1168	1091	1047	987
Testing	415	412	316	308	301	295	292	272	261	246
Accuracy [%]	73.67	90.46	84.22	83.08	90.92	80.85	83.44	90.97	87.54	88.80
User ID	21	22	23	24	25	26	27	28	29	30
Training	984	969	948	920	920	840	835	817	810	731
Testing	246	242	237	229	230	209	208	204	202	182
Accuracy [%]	82.96	83.26	81.97	71.10	80.37	79.10	80.62	87.08	87.60	79.19
User ID	31	32	33	34	35	36	37	38	39	40
Training	715	705	705	703	686	670	664	643	1303	604
Testing	178	176	176	175	171	167	166	160	325	151
Accuracy [%]	79.04	88.11	74.16	94.25	76.32	86.47	92.65	84.94	72.86	87.57
User ID	41	42	43	44	45	46	47	48	49	50
Training	588	579	573	567	544	500	479	474	470	446
Testing	147	144	143	141	136	125	119	118	117	111
Accuracy [%]	86.79	80.73	86.93	84.86	86.09	91.97	92.07	86.35	88.92	85.18
User ID	51	52	53	54	55					
Training	440	621	637	885	1267					
Testing	110	155	159	221	316					
Accuracy [%]	81.53	76.91	80.47	87.48	85.23					

be trained independently of each other. To speed up the learning of the discriminator, we use data from the autoencoder training instead of computing the encoder output again. Otherwise, only 40 users could be trained on the aforementioned equipment without using this technique. To train more users, a machine with more GPU memory is required. This limitation, however, was not valid after training and if we had a machine with more memory only for training, it would be possible to use the trained system on the equipment we had at our disposal. Another indicator is the number of URL sessions processed by the system in a specific time. Using the above-mentioned GPU, we are able to process a set (mini-batch) of 200 sessions in 36.8 ms per package, which shows that the system can be used in real-time scenarios. This is the number of sessions that can be loaded at once on a GPU with 11 GB memory.

5 Conclusion

Our system based on autoencoders and one-class CNN, is a new approach to system security and anomaly detection in user behaviour. It provides continuous authentication of computer network users by software interaction analysis. We used real text data from the network traffic instead of hand-crafted traffic statistics as in the case of the previous approaches. Moreover, the proposed framework is a universal anomaly detection system applied in the paper for user authentication. We proposed three architectures that differ in size and complexity. The best architecture presented in the paper allows to add and remove any user without having to retrain the whole system. Thanks to this, we can save both time and computational resources. The presented solution can be used for other behavioural security solutions to create user profiles utilizing other available data. Future research would involve alternate methods to create autoencoders to improve accuracy. In the presented article, we used the same way of training the autoencoder for each user. To obtain better results, it would be beneficial to create a dedicated autoencoder architecture for each user, but this would involve a change in the implementation of the discriminator.

Acknowledgement. The project financed under the program of the Polish Minister of Science and Higher Education under the name “Regional Initiative of Excellence” in the years 2019–2022 project number 020/RID/2018/19, the amount of financing 12,000,000.00 PLN.

References

1. Blum, A., Wardman, B., Solorio, T., Warner, G.: Lexical feature based phishing URL detection using online learning. In: Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security, pp. 54–60. ACM (2010)
2. Boles, A., Rad, P.: Voice biometrics: deep learning-based voiceprint authentication system. In: 12th System of Systems Engineering Conference (SoSE), pp. 1–6, June 2017

3. Buriro, A., Crispo, B., Delfrari, F., Wrona, K.: Hold and sign: a novel behavioral biometrics for smartphone user authentication. In: IEEE Security and Privacy Workshops (SPW), pp. 276–285, May 2016
4. Buriro, A., Crispo, B., Conti, M.: AnswerAuth: a bimodal behavioral biometric-based user authentication scheme for smartphones. *J. Inf. Secur. Appl.* **44**, 89–103 (2019)
5. Cao, Y., Liu, B., Long, M., Wang, J.: HashGAN: deep learning to hash with pair conditional Wasserstein GAN. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1287–1296 (2018)
6. Chalapathy, R., Menon, A.K., Chawla, S.: Anomaly detection using one-class neural networks. arXiv preprint [arXiv:1802.06360](https://arxiv.org/abs/1802.06360) (2018)
7. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. *J. Am. Soc. Inform. Sci.* **41**(6), 391–407 (1990)
8. Goodfellow, I., et al.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)
9. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint [arXiv:1502.03167](https://arxiv.org/abs/1502.03167) (2015)
10. Kwon, D., Natarajan, K., Suh, S.C., Kim, H., Kim, J.: An empirical study on network anomaly detection using convolutional neural networks. In: IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pp. 1595–1598. IEEE (2018)
11. Mahfouz, A., Mahmoud, T.M., Eldin, A.S.: A survey on behavioral biometric authentication on smartphones. *J. Inf. Secur. Appl.* **37**, 28–37 (2017)
12. Ronneberger, O., Fischer, P., Brox, T.: U-Net: convolutional networks for biomedical image segmentation. In: Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F. (eds.) MICCAI 2015. LNCS, vol. 9351, pp. 234–241. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24574-4_28
13. Salakhutdinov, R., Hinton, G.: Semantic hashing. *Int. J. Approximate Reasoning* **50**(7), 969–978 (2009). Special Section on Graphical Models and Information Retrieval
14. Shi, W., et al.: Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1874–1883 (2016)
15. Sun, Y., Chen, Y., Wang, X., Tang, X.: Deep learning face representation by joint identification-verification. In: Advances in Neural Information Processing Systems, pp. 1988–1996 (2014)
16. Xiao, G., Milanova, M., Xie, M.: Secure behavioral biometric authentication with leap motion. In: 4th International Symposium on Digital Forensic and Security (ISDFS), pp. 112–118, April 2016
17. Zhang, P., You, X., Ou, W., Chen, C.P., Cheung, Y.M.: Sparse discriminative multi-manifold embedding for one-sample face identification. *Pattern Recognit.* **52**, 249–259 (2016)
18. Zhang, X., LeCun, Y.: Byte-level recursive convolutional auto-encoder for text. arXiv preprint [arXiv:1802.01817](https://arxiv.org/abs/1802.01817) (2018)
19. Zhang, X., Zhao, J., LeCun, Y.: Character-level convolutional networks for text classification. In: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1. NIPS 2015, pp. 649–657. MIT Press, Cambridge (2015)
20. Zouina, M., Outtaj, B.: A novel lightweight URL phishing detection system using SVM and similarity index. *Hum. Cent. Comput. Inf. Sci.* **7**(1), 17 (2017)