Code Protection Techniques when Distributed in Source Format: an Adobe Connect Pod Written in Javascript

Alessandro Simonetta[1], Francesco Rinaldi[1]

[1]Department of Enterprise Engineering, University of Rome "Tor Vergata", Via del Politecnico n.1, 00133, Rome, Italy

Abstract

The purpose of this article is to describe techniques for protecting code when distributed in source format. This situation occurs when, for instance, the client component of a web application, whose source code is easily extractable from the browser even by inexperienced users. The case study proposed uses the Adobe Connect© platform, an emerging technology in the field of video communication, content sharing and e-learning environments, which allows to easy integration of applications written in javascript language. The astonishing ease of realization of embedded applications within Adobe®'s ecosystem contrasts with the impossibility of protecting the work done, which is visible and redistributable simply by copying the file containing it. The unwary author may thus run the risk of seeing his work thwarted by losing any intellectual property rights arising from the use of the software he has created. For this reason we have realized a form of intellectual property protection when software is distributed in source format.

Keywords

development, coding, source code, software protection, javascript, adobe connect, copyright, API, COVID-19, Pod, intellectual property

1. Introduction

Software production has evolved considerably in recent years thanks to the advent of new technologies, innovative development and deployment methodologies, such as *DevOps* [1][2][3]. Monolithic applications, which are often obsolete, have been replaced by microservice applications with greater advantages in terms of resilience, scalability, speed of development (*time-to-market* and *continuous integration & delivery*) and, last but not least, simplicity of release on the cloud [4][5]. The availability of server-side microservices has made the front-end graphical interfaces on the clients strongly decoupled from the rest of the code. This facilitated integration via *Application Programming Interface* (API) with the software platforms of *Content Management System* (CMS), *Customer Relationship Management* (CRM) and and *Learning Management System* (LMS). In most cases these integrations, Pods or Plug-ins, are limited to the creation of a front-end application that interacts with the exposed services. In this context, the protection of the produced source code has become a complex issue to deal with and difficult to manage.

This article will briefly review the history of software application protection systems. It is followed by the proposed solution to protect the source code of a web application. Finally, a real case will be described, i.e. the protection of a Pod realized in javascript on Adobe Connect©[1].

2. History of software protection systems

The protection of developed code is a problem that has always existed in the field of information technology. Source code written in compiled programming languages (e.g. C++) is transformed into object code, directly comprehensible by the machine. This step makes the code unintelligible to a human being because it is coded in binary. However, it is always possible to use a decompiler that allows you to restore it to a source form similar (but not the same) to the original [6]. Creating an application that cannot be cracked is not an easy task. To understand the extent of the phenomenon, just consult the international reports [7],[8]. When it comes to protection or security, we know that it is almost impossible to use absolute terms, but it is necessary to use relative criteria. Indeed, what we have to do is study the motivations, the level of preparation and the financial resources of those who might be interested in compromising the protection of our software. A general criterion is to assume a higher position than the potential positions of the other parties, because it would not make sense to spend more energy than that. In the assessment it must be considered that the free circulation of software, even if unauthorised, favours its dissemination and knowledge [9][10], that which is normally paid for by investments in marketing campaigns. On the other hand, if a company considers

[1]https://www.adobe.com/products/adobeconnect

SYSTEM 2021 @ Scholar's Yearly Symposium of Technology, Engineering and Mathematics. July 27–29, 2021, Catania, IT

alessandro.simonetta@gmail.com (A. Simonetta);
franin@gmail.com (. F. Rinaldi)
0000-0002-0877-7063 (A. Simonetta)

software useful, it will certainly find it more convenient to purchase it than to risk using it without a license. Over time, various techniques have been used to protect software applications, methods that have changed with the evolution of architectures and, above all, with the advent of the network that has made it possible to control licenses in real time, initially on a LAN and later on the Internet.

The first local techniques used a protection which checks whether the license related to the hardware on the machine where it was first installed. This required a double step: the generation of a key depending on the hardware factors of the machine (serial number of the hard disk, date of the ROM, MAC address of the network card, ...) at the time of installation and the verification, at each execution, that the key calculated at that moment was the same as the one registered during installation. Clearly, when there was a failure of a hardware component, the installation had to be repeated in order to restore the correct values. To overcome this problem, the alternative was the availability of a token stored on non-copyable removable media. Early systems used common floppy disks, later CD-ROMs, which had defective sectors in some tracks of the medium. In this way, it was not possible to duplicate the medium with the faulty tracks because the copying programs only acted on the data. Soon, copiers were created that were able to mark the faulty tracks, so that the media could be reproduced identical to the original, making it possible to use the software on several machines (each with duplicate protection media). A better protection was only introduced later with the advent of USB memories, both because these hardware devices are difficult to duplicate, and also because they are only accessible from the relevant application. The protection provided by a USB device, commonly called *dongle*, requires the use of a key at the running location, so if there are several users using the software at different times, they must be able to exchange the device. It is also possible associates the license to a person, who carries the device with him and can therefore use the software on different locations. Obviously, in order to solve the difficulties caused by the use of a dongle, it would be sufficient to purchase several user licenses.

With the increasing use of networks, computers were no longer stand-alone (i.e., isolated from each other) but they could exchange information with each other. At this point, the token could be used by a central server that had the task of controlling the installation of applications within the local network on the basis of the license purchased, distributing the privileges of use (*license manager*). The need for protection has diversified in web architectures, especially in those for which is sufficient to have access in terms of credentials on a remote server, while for copying the code, it is distributed in open format (typically HTML and javascript) on the client [11]. The availability of the source code renders all the protection systems examined so far useless, since a malicious user could easily remove them.

Although a developer can always register the source code to be able to legally claim both authorship and wrongful use, none of these threats are better deterrents than a well-designed protection system.

3. The proposed solution

The proposed solution considers the source code distributed on the client of a typical web application as the target to be protected. It is based on three concentric protection levels:

- the basic level, it marks the GUI of the application with the logo and the name of the licensee (*watermark*);
- the intermediate level, it makes the source code unreadable and unmodifiable to a programmer through code obfuscation techniques;
- the server level (license manager):
  - monitors the clients and decides action strategies in relation to the client's license;
  - sends the missing source code to the client;
  - checks that the client's code has not been altered and that the logo and the licensee's name are consistent with the license.

The architecture of the proposed solution is shown in the Fig. 1.

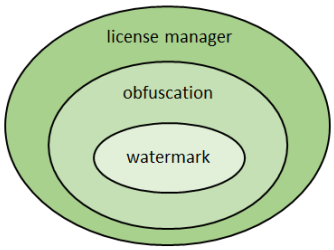

Figure 1: The architecture of the proposed solution

3.1. Basic level: watermarking

In order to prevent the diffusion and use of counterfeit copies of a product (not necessarily software), a good deterrent may be to indelibly and clearly mark the product itself with the logos and name of the licensee. It is unlikely that a professional user will use it with the name of another licensee, also because if there were a check

on the software, it would be difficult to prove that the purchase was made legally.

Ideally, a certificate of originality should be included in the HTML file, or in the javascript source, without the ability to be deleted or altered, the same level of guarantee that the watermark on a banknote has that ensures its originality [12]. On the other hand, in a source code, it is quite easy to alter the loading of an image from an external file or to change the name of the licensee if it is written in plain text as a sequence of characters. The solution is to devise an encapsulation and information hiding mechanism that makes the licensee's logo and name unreadable in the HTML file but visible to *runtime*, only after a complex calculation process. This first level of protection must be followed by two others, which aim to encapsulate and protect it. We can say that it is similar to the technique that has been used for centuries to protect fortresses with outer walls.

3.2. Intermediate level: code obfuscation

The second level of protection is based on the technique of source code obfuscation, a practice well known in literature [13][14].

In recent years, we are witnessing machine learning being used in a wide variety of ways [15][16][17][18] thanks to the discovery of increasingly efficient implementations [19][20][21]. It has been shown that such algorithms can also be successfully implemented in obfuscation techniques [22][23].

However, software obfuscation for the purpose of intellectual property protection remains a very challenging topic [24], even though it has been shown that, while reading a web page, it is possible to automatically detect the content of obfuscated javascript strings [25]. The transformation of the code and its execution flow is isofunctional: the original behavior is kept unchanged. What changes is the complexity, which increases because processes are made convoluted and variables are scattered throughout the code. The goal is to transform a source code and make it similar to an object code, from the point of view of comprehensibility. There are several tools that perform this transformation, but there are just as many that perform the reverse operation (*deoffuscators*)[26][27]. Although the final result is always far from the original one, it is possible to insert in the source code some useless instructions (*junk code*) that will never be used and that have the purpose only to amplify the complexity. As an example, let's see how an obfuscator acts on a simple javascript function:

```
function localStorageincrement(){\\
    localStorage.setItem('streamL',
        (parseInt(localStorage.
          getItem('streamL'))+1)+'');
    localStorage.
        setItem('w','&w=n');
}
```

After applying obfuscation techniques:

```
function localStorageincrement(){
var _0x3f436f=_0x30e3d7;localStorage[
'setItem'](_0x3f436f(0x166),
parseInt(localStorage[_0x3f436f(0x12e)]
('streamL'))+0x1+''),localStorage
[_0x3f436f(0x176)]('w',_0x3f436f(0x17b));}
}
```

The transformation proves the difficulty of interpretation that an attacker might have in deducing the behavior of the function in question. This difficulty is amplified if the code is very long and, above all, if there are useless parts to analyze. In this case the reverse engineering activity is long, laborious and has little chance of success.

3.3. Server level: license manager

The third level of protection is by means of a remote server: the *license manager* which monitors who uses the client, sends the parts of code that the client lacks in order to work, and checks which there are no alterations to the code (e.g. substitution of the logo or name of the licensee). Therefore, the application that uses this protection system needs a connection to the Internet in order to work. This requirement is also necessary to guarantee the functionality of the application itself since it is based on web technology on an internet network.

Moreover, the same server has modifiable policies where the application usage criteria are defined. According to the policies of software diffusion and to the risk (loss of profit, illicit duplication,...) you want to assume in maintaining active functioning demonstrative licenses, you can decide if:

- the demonstration state is unlimited and used to advertise the product;
- the demo status remains active for a limited trial period, after which if there is no connection to the server, the license expires and stops working;
- the product does not work if it has no connection with the server.

Whenever the client software is started, it connects with the server and provides information about the tasks it is called to perform. This conversation is necessary for the client to have all the software necessary for its operation.

Generally, the minimum requirement for requesting services from a server is to be authenticated. In the absence of authentication, an attack from a not trusted

client can be avoided by excluding the possibility of the *cross-domain*: the source code will be sent only if the request comes from an authorized client and domain.

The availability of the license manager is also fundamental to check if the javascript file on the client still contains the logos of the registered licensee or if some form of code alteration has happened. All this can be easily implemented with a hashing function, so if a user should manage to penetrate the first two protection levels and change the logo in the source code, the hashing function will return a different value from the expected one, and the license manager could decide the best strategy to implement.

Any software application that wants to adopt the proposed protection solution does not need to know not send personal data in the conversation from the client to the license manager. Also because it depends on the regulations in the country where the user is located and the license manager server too. In Europe such legislation is Regulation (EU) 2016/679 (General Data Protection Regulation)[28].

4. Case Study

This Case Study aims to demonstrate, with a practical example, the theoretical concepts described so far. In recent years we have been witnessing an increase in the use of video communication software especially in relation to the problem of the pandemic caused by the coronavirus, SARS-CoV-2, also known as COVID-19 [29][30][31][32]. Similarly, new requirements have arisen encouraging the distancing of people in all meeting occasions in social and work occasions [33], for example, in reserving seats in the cafeteria of a work environment or in the need to perform tasks remotely through a collaborative platform for meetings, training or job interviews.

These platforms can be enriched with new functionalities and promote new job opportunities for programmers who have a new space where to spread their ideas. For this reason, we have chosen to use the Adobe Connect© communication platform for reasons of dissemination [34][35][36] but also because it allows us to write applications in a programming language already known and established as javascript.

4.1. The Adobe Connect platform

Adobe Connect© makes available its *Software Development Kit* (SDK)[2] that contains all the documentation and tools useful for developers to build embedded applications. The SDK consists of:

- the manual for the use of the javascript-capable Application Programming Interface (API);
- the example application to start from;
- the library itself that contains the classes and methods to be called upon in development.

The application is assembled in a container that we will call Pod in ZIP format. The process of developing and deploying an application (Fig. 2) consists of the following steps:

- collection of functional requirements including the definition of the GUI;
- definition of the layout of the web page;
- identification of the API necessary for the functioning of the application;
- development of the application components (custom library);
- release of the ZIP file on the Adobe Connect© server.

The collection of requirements is preparatory to the design of the human-machine interface (HTML file) and to identify the APIs necessary for the operation of the application through the mechanism of *callback*. In this way it will be possible to activate the new custom developed functions, following the events that the system will receive. Once the application components have been created, they will be inserted in the javascript library accessible by the HTML file. At this point it will be possible to create the ZIP file that will contain:

- the HTML file with the page of the developed application;
- the configuration file (*breeze-manifest.xml*) with the names and paths of the application components;
- the folder *lib* with the SDK and the custom libraries developed;
- the folder *css* eventually added for the webpage layout settings.

To distribute the Pod it will be sufficient to load it into Adobe Connect© and use the product sharing mechanisms (e.g. virtual rooms) without the need for installation and configuration.

Below there is a simple example of using the SDK classes available in the javascript file (*connect_customPodSDK.js*):

```
<script type="text/javascript">
 cpu=ConnectCustomSDK.SyncConnector||{};
 cpu.init(onConfigured,
      "com.adobe.connect.basiclistsync",
      "9.5.001","connectsdkhook");
</script>
```

[2] https://console.adobe.io/servicesandapis

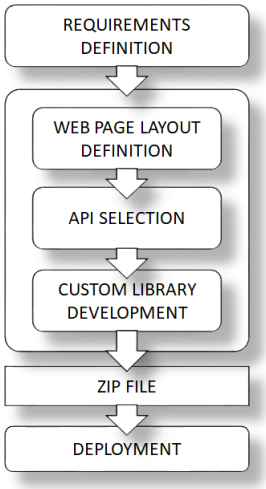

Figure 2: Pod development and deply process

4.2. The development of a protected Pod

Suppose we want to create an application that allows a teacher to show, during a teaching session (meeting), a video presentation on a streaming server available on the Internet.

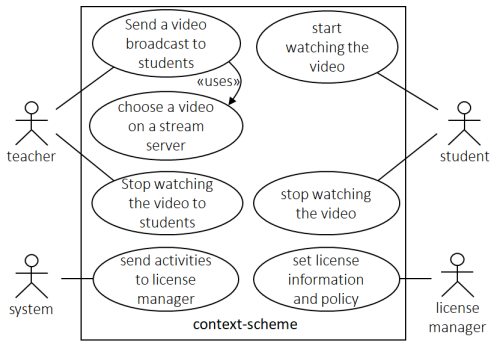

Figure 3: The UML context-scheme

The application must have two modes of operation depending on the user who is connected (teacher or learner). The teacher must be able to choose the video to send in broadcast to all students, start it and block it. The student must view the video sent by the teacher during the teaching session.

The application sends information to the license manager about the activities and receives the code parts and details about the operation mode from the license manager. In Fig. 3 is shown in UML language the context diagram of the application.

The class *MyUserDetail* available in the SDK allows us

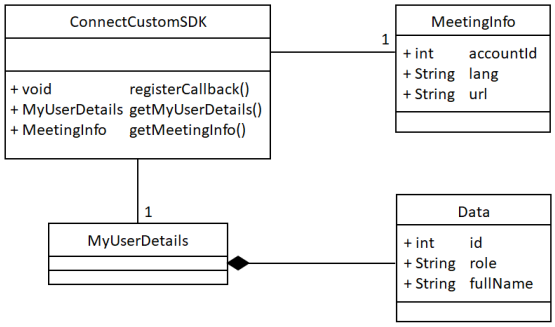

Figure 4: UML Class Diagram ConnectCustomSDK

to read the information related to the user role (Fig. 4) and consequently to select the operational mode.

```
var myUserData = cpu.getMyUserDetails();
if(myUserData.data.role=='owner') {
    // teacher actions
}else{
    // student actions
}
```

Once the application logic has been defined, it will be necessary to link the system events to the custom javascript methods through the callback registration. For simplicity and without loss of generality, we will define javascript methods that are homonyms to the Adobe Connect© API. Communication between the teacher and the learner is done using the APIs: *dispatchSyncMessage* and *syncMessageReceive*. The first API send broadcast messages from one participant to the others, the second API allows participants to receive messages.

```
cpu.registerCallback("UserJoined",
                                    UserJoined);
//New user joins the room
function UserJoined(evt){
    // evt.user is istance of MyUserDatails
    var fullname=evt.user.fullname;
    // send broadcast the message
    cpu.dispatchSyncMessage("USERJOINED",
                            [fullname],
                            false,true);
    // synchronizes the video for new user
    checkVideoSync();
}
```

During a teaching session, a teacher may need to stop the video to add a contribution relating to the video they have just watched. In this case, it is useful to realize a function that allows the video to be blocked for all students following the lesson. This can be easily done sending a message to all learner connected:

```
cpu.registerCallback
     ("syncMessageReceived",
                    syncMessageReceived);

function syncMessageReceived(syncMsg){
     if(syncMsg.msgNm=='STOPVIDEO')
               stopVideo();
}
```

In Fig. 5 it is shown how the Pod is distributed from the Adobe Connect© server to the student client workstations. This is done automatically with no intervention required from the user connecting to the meeting.

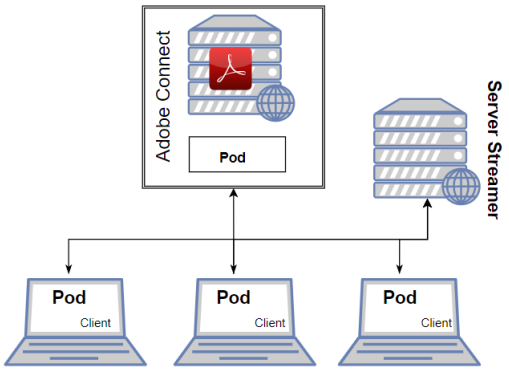

Figure 5: Pod distribution to learner's clients

Now let's see how the authorised client requests from the license manager the javascript code it needs to function. One method is to perform a GET call from the client to the server that returns the code inserted in a client's HTML tag (*demo* in the following example).

```
<p id="demo"></p>
....
\$(document).ready(function() {
  \$.ajax(
    {
       url:"https://servername/inject",
      'method': 'GET',
      'success': function(answer){
            document.getElementById("demo")
            .innerHTML = answer[0].script;
      },
      'error':function(){
         alert('something wrong');
       }
     ....
```

5. Conclusion

The software intellectual property's protection is a complex issue. When the code is distributed in the compiled form, it maintains an intrinsic basic protection due to the fact that there is no visibility of the processes wired into it. Although, it is always possible to trace the source code that generates it, the source code (except in the case of particular programming languages) will never have the same readability and form as the original code. Therefore, it may be more difficult to copy it and reuse it in environments other than where it was licensed.

In the case of open source distribution, there are various forms of licenses that the owner can choose, but it is not easy to block copying and use.

The growing use of video communication, content sharing and e-learning environments is encouraging the development of embedded applications and offering new scenarios and opportunities for work.

The Adobe Connect© communication platform encourages the development of applications in the javascript language that is already known to the developer community.

The idea of the proposed solution shows how it is possible to adopt a multi-layered protection solution in order to protect the intellectual property of the developed code when it is distributed in source format.

Although the case study is focused on an embedded application, the proposed method remains valid in general and can be adopted regardless of the programming language.

6. Acknowledgments

We would like to thank Luciano Fazio and Katherine L. Ryan for their careful revisions and valuable suggestions to the text. We are also grateful to Maria Cristina Paoletti and Emanuele Iannaccone for the stimulating discussion on the research articles and to the President of the UNI CT 510 Security Commission, Fabio Guasconi, for his interesting suggestions on security issues.

References

[1] P. Perera, R. Silva, I. Perera, Improve software quality through practicing DevOps, 2017, pp. 1–6. doi:10.1109/ICTER.2017.8257807.

[2] M. Senapathi, J. Buchan, H. Osman, Devops capabilities, practices, and challenges: Insights from a case study, in: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, EASE'18, Association for Computing Machinery, New York, NY, USA, 2018, p. 57–67. URL: https://doi.org/10.1145/3210459.3210465. doi:10.1145/3210459.3210465.

[3] C. Napoli, G. Pappalardo, E. Tramontana, Using modularity metrics to assist move method refactoring of large systems, in: 2013 Seventh International Conference on Complex, Intelligent, and Software Intensive Systems, IEEE, 2013, pp. 529–534.

[4] S. R. Dileepkumar, J. Mathew, Optimize continuous integration and continuous deployment in azure DevOps for a controlled microsoft .NET environment using different techniques and practices, IOP Conference Series: Materials Science and Engineering 1085 (2021) 012027. URL: https://doi.org/10.1088/1757-899x/1085/1/012027. doi:10.1088/1757-899x/1085/1/012027.

[5] D. Taibi, V. Lenarduzzi, C. Pahl, Continuous Architecting With Microservices and DevOps: a Systematic Mapping Study, 2019. doi:10.1007/978-3-030-29193-8_7.

[6] O. Katz, Y. Olshaker, Y. Goldberg, E. Yahav, Towards neural decompilation, 2019. arXiv:1905.08325.

[7] United States Trade Representative (USTR), Special 301 report, 2021. URL: https://ustr.gov/sites/default/files/files/reports/2021/2021%20Special%20301%20Report%20(final).pdf.

[8] S. Sahni, I. Gupta, Piracy in the Digital Era: Psychosocial, Criminological and Cultural Factors, 2019. doi:10.1007/978-981-13-7173-8.

[9] J. Wang, R. L. Axtell, A. Loerch, Utilizing the positive impacts of software piracy in monopoly industries (2017). URL: https://dl.acm.org/doi/10.5555/3106078.3106083.

[10] A. Prasad, V. Mahajan, How many pirates should a software firm tolerate? an analysis of piracy protection on the diffusion of software, International Journal of Research in Marketing 20 (2003) 337–353. doi:10.1016/j.ijresmar.2003.02.001.

[11] T. Groß, T. Müller, Protecting javascript apps from code analysis, in: Proceedings of the 4th Workshop on Security in Highly Connected IT Systems, SHCIS '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 1–6. URL: https://doi.org/10.1145/3099012.3099018. doi:10.1145/3099012.3099018.

[12] L. Regano, D. Canavese, C. Basile, A. Lioy, Towards optimally hiding protected assets in software applications, in: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), 2017, pp. 374–385. doi:10.1109/QRS.2017.47.

[13] S. Hosseinzadeh, S. Rauti, S. Laurén, J.-M. Mäkelä, J. Holvitie, S. Hyrynsalmi, V. Leppänen, Diversification and obfuscation techniques for software security: A systematic literature review, Information and Software Technology 104 (2018) 72–93. URL: https://www.sciencedirect.com/science/article/pii/S0950584918301484. doi:https://doi.org/10.1016/j.infsof.2018.07.007.

[14] C. K. Behera, D. L. Bhaskari, Different obfuscation techniques for code protection, Procedia Computer Science 70 (2015) 757–763. URL: https://www.sciencedirect.com/science/article/pii/S1877050915032780. doi:https://doi.org/10.1016/j.procs.2015.10.114, proceedings of the 4th International Conference on Eco-friendly Computing and Communication Systems.

[15] G. Capizzi, G. Lo Sciuto, C. Napoli, E. Tramontana, A multithread nested neural network architecture to model surface plasmon polaritons propagation, Micromachines 7 (2016). doi:10.3390/mi7070110.

[16] R. Avanzato, F. Beritelli, M. Russo, S. Russo, M. Vaccaro, Yolov3-based mask and face recognition algorithm for individual protection applications, in: CEUR Workshop Proceedings, 2020, pp. 41–45.

[17] G. Capizzi, G. Lo Sciuto, C. Napoli, E. Tramontana, M. Woźniak, A novel neural networks-based texture image processing algorithm for orange defects classification, Int. J. Comput. Sci. Appl. 13 (2016) 45–60.

[18] C. Napoli, F. Bonanno, G. Capizzi, Exploiting solar wind time series correlation with magnetospheric response by using an hybrid neuro-wavelet approach, Proceedings of the International Astronomical Union 6 (2010) 156–158. doi:10.1017/S1743921311006806, cited By 26.

[19] G. C. Cardarilli, L. D. Nunzio, R. Fazzolari, D. Giardino, A. Nannarelli, M. Re, S. Spanò, A pseudo-softmax function for hardware-based high speed image classification, Scientific Reports 11 (2021). doi:10.1038/s41598-021-94691-7.

[20] S. Spanò, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Matta, A. Nannarelli, M. Re, An efficient hardware implementation of reinforcement learning: The q-learning algorithm, IEEE Access 7 (2019) 186340–186351. doi:10.1109/ACCESS.2019.2961174.

[21] S. Russo, S. Illari, R. Avanzato, C. Napoli, Reducing the psychological burden of isolated oncological patients by means of decision trees, volume 2768, 2020, pp. 46–53.

[22] M. Romanelli, K. Chatzikokolakis, C. Palamidessi, Optimal obfuscation mechanisms via machine learning, arXiv preprint arXiv:1904.01059 (2019).

[23] D. Canavese, L. Regano, C. Basile, A. Viticchié, Estimating software obfuscation potency with artificial neural networks, in: G. Livraga, C. Mitchell (Eds.), Security and Trust Management, Springer International Publishing, Cham, 2017, pp. 193–202.

[24] S. Schrittwieser, S. Katzenbeisser, J. Kinder, G. Merzdovnik, E. Weippl, Protecting software through obfuscation: Can it keep pace with

progress in code analysis?, ACM Comput. Surv. 49 (2016). URL: https://doi.org/10.1145/2886012. doi:10.1145/2886012.

[25] Y. Choi, T. Kim, S. Choi, C. Lee, Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis, in: Y.-h. Lee, T.-h. Kim, W.-c. Fang, D. Ślęzak (Eds.), Future Generation Information Technology, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 160–172.

[26] Y. Fang, C. Huang, Y. Su, Y. Qiu, Detecting malicious javascript code based on semantic analysis, Computers & Security 93 (2020) 101764. URL: https://www.sciencedirect.com/science/article/pii/S0167404820300481. doi:https://doi.org/10.1016/j.cose.2020.101764.

[27] B. Yadegari, B. Johannesmeyer, B. Whitely, S. Debray, A generic approach to automatic deobfuscation of executable code, in: 2015 IEEE Symposium on Security and Privacy, 2015, pp. 674–691. doi:10.1109/SP.2015.47.

[28] European Union, Regulation 2016/679 (General Data Protection Regulation), 2016. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679.

[29] K. A. Karl, J. V. Peluchette, N. Aghakhani, Virtual work meetings during the covid-19 pandemic: The good, bad, and ugly, Small Group Research (2021) 10464964211015286. URL: https://doi.org/10.1177/10464964211015286. doi:10.1177/10464964211015286.

[30] Z. R. Alashhab, M. Anbar, M. M. Singh, Y.-B. Leau, Z. A. Al-Sai, S. Abu Alhayja'a, Impact of coronavirus pandemic crisis on technologies and cloud computing applications, Journal of Electronic Science and Technology 19 (2021) 100059. URL: https://www.sciencedirect.com/science/article/pii/S1674862X20300665. doi:https://doi.org/10.1016/j.jnlest.2020.100059, special Section on In Silico Research on Microbiology and Public Health.

[31] M. H. Nguyen, J. Gruber, J. Fuchs, W. Marler, A. Hunsaker, E. Hargittai, Changes in digital communication during the covid-19 global pandemic: Implications for digital inequality and future research, Social Media + Society 6 (2020) 2056305120948255. URL: https://doi.org/10.1177/2056305120948255. doi:10.1177/2056305120948255, pMID: 34192039.

[32] J. Hacker, J. vom Brocke, J. Handali, M. Otto, J. Schneider, Virtually in this together – how web-conferencing systems enabled a new virtual togetherness during the covid-19 crisis, European Journal of Information Systems 29 (2020) 563–584. URL: https://doi.org/10.1080/0960085X.2020.1814680. doi:10.1080/0960085X.2020.1814680.

[33] K. Kaspar, Motivations for social distancing and app use as complementary measures to combat the covid-19 pandemic: Quantitative survey study, J Med Internet Res 22 (2020) e21613. URL: http://www.jmir.org/2020/8/e21613/. doi:10.2196/21613.

[34] B. Jamalpur, Kafila, K. R. Chythanya, K. S. Kumar, A comprehensive overview of online education – impact on engineering students during covid-19, Materials Today: Proceedings (2021). URL: https://www.sciencedirect.com/science/article/pii/S2214785321008464. doi:https://doi.org/10.1016/j.matpr.2021.01.749.

[35] A. A. Oloyede, N. Faruk, W. O. Raji, Covid-19 lockdown and remote attendance teaching in developing countries: A review of some online pedagogical resources, African Journal of Science, Technology, Innovation and Development (2021) 1–19. URL: https://doi.org/10.1080/20421338.2021.1889768. doi:10.1080/20421338.2021.1889768.

[36] S. Caliskan, R. A. Kurbanov, R. I. Platonova, A. M. Ishmuradova, D. G. Vasbieva, I. V. Merenkova, Lecturers views of online instructors about distance education and adobe connect, International Journal of Emerging Technologies in Learning (iJET) 15 (2020) 145–157. URL: https://online-journals.org/index.php/i-jet/article/view/18807.