2019 22nd International Conference on Control Systems and Computer Science (CSCS)

Data Loss Prevention and Data Protection in Cloud Environments based on Authentication Tokens

Vlad Bucur, Ovidiu Stan, Liviu Miclea
Department of Automation, Technical University of Cluj Napoca, Cluj, Romania
vbucur1@gmail.com, ovidiu.stan@aut.utcluj.ro, liviu.miclea@aut.utcluj.ro

***Abstract*— Data leek issues represents a very high or critical importance within the cloud developers. Given the tremendous growth of cloud computing in the past 3 years the authors of this paper believe that an analysis of the implementation of modern security solutions for data leaks and data protection is of the utmost importance. Authentication tokens, provided by several different cloud providers and third parties, are one of the most common and useful tools that developers have in order to secure their applications. The goal of this paper is to analyze how these tokens are currently used and to provide a solution in order to improve their implementation by reviewing their compatibility with novel security concepts or challenges such as data tagging, hybrid data security algorithms, microservices deployment using Docker or serverless applications.**

Keywords*— *security token; web token; access token; ID token; multi-cloud storage; multi-cloud security; computing resource sharing; data-tagging, data loss protection, security audit.

I. Introduction

The starting point of this research paper was representing by the results publish by the Velumadhava et. all. in 2015 [1]. According to their conducted study on cloud security, 88% of all data leak issues were considered of Very High or Critical importance to cloud developers. Issue related to data segregation and protection were of an even higher criticality, as a staggering 92% of reported problems were considered of Critical or Very High priority.

In the ever-evolving field of cybersecurity tokens themselves have seen major changes of the past few years. A token isn't just a hardware device any more, but can also be credentials used to access an API. These tokens, known as Access Tokens, are simply opaque strings or JSON web tokens that do not require any device to generate them, rather they can be generated by developers and sent in an HTTP Authorization header directly to the API for authentication.

Given how popular tokens are, and how likely they are to be in the future, the authors of this paper believe that an in-depth analysis and implementation study should be conducted on how to maximize the use of tokens in securing file transfers and in order to prevent data loss. Data loss and data protection aren't mutually exclusive terms and, despite an initial inclination to see these terms as related to hardware failures rather than malicious intent, it must be specified that data is just as likely to be compromised by attacks.

Therefore, this paper will run through a quick overview of how tokens work in cyber security, with a particular focus on web tokens and operation in the cloud. Then the authors will analyze token functionality in file transfers and authentication for file transfers in the cloud, including a deep-learning mechanism for identify security critical files, before moving on to analyzing an implementation of tokens in vendor free cloud environments simply known as “multi-clouds”. Finally, the paper will address the potential challenges involved in implementing our approach.

II. The use of Tokens in Cyber-Security

In terms of cyber security tokens are used in two-factor-authentication where a user has a personal pin number that they input into a device which then generates a code they can use in order to log in. The focus of this paper though will be on tokens that are used in securing access to web-based APIs.

A token can be generated using specific algorithms in various sizes ranging from 32 bits to thousands of bits. The gains from generating a token that is, for example, 2048-bit though are very slight. Current computer technology considers a 256-bit token secure for centuries. It's important to note that an even further layer of security is usually added on top of the generated token. This is usually done by using a Secure Sockets Layer, SSL for short, that keeps an internet connection secure and safeguards the data sent through it.

There are multiple token vendors, but for the rest of this paper we will be referring to tokens generated by Auth0, a major token provider. Auth0 has 4 main categories of distinct tokens [2]: *ID tokens, Access tokens, Identity provider access tokens* and *Refresh tokens*. Tokens are generated as a seemingly senseless piece of code but once interpreted it contains the login or personal information of a user and an expiration date or time for the token [3]. The expiration time of the token is as crucial to its meaning as is its obfuscation of user information. Without an expiration time a token can be reused by anyone that gets access to it, including malicious parties.

ID tokens are actually a JSON strings that contains the user profile attributes represented in the form of claim [4]. The token is consumed by the application in order to get the user's email, user name, phone and so forth. Most tokens in general, and ID tokens in particular, use JSON Web Tokens (JWTs), because these are a compact, URL-safe means of sending strings [2]. They can be easily sent through a POST call to the API or inside an HTTP header. Furthermore, they are completely self-contained as they have all the required information about the entity and thus avoid querying the database more than once [2][7].

Access tokens are perhaps the ones that are most familiars to both users and developers. They are the pieces of code that inform the API that the bearer of the token has the authorization to be accessing that data [3]. These tokens can use both JWTs and obfuscated code. In the same vein as access tokens are identity provider access tokens (IPATs). These latter tokens are provided by a third party but that can be used interchangeably by the user when logging in to a particular API [5]. The best example of a scenario where IPATs are used would be when logging in to Facebook and then using Facebook authentication to login to another 3rd party website. In this case the issuing provider validates the

2379-0482/19/$31.00 ©2019 IEEE
DOI 10.1109/CSCS.2019.00128

token and the renewal is left up to the third party: either pass it back to Facebook for authenticating or authenticate it server side [5]. Finally, there are refresh tokens which contain the required information to obtain a new access token or ID token [6].

III. Token Functionality in File Transfers

Currently, the most common means of securing files for transfer is to use a generated access token to secure the transfer of the file itself. These tokens are used to basically handshake the application that is requesting the file and then said file is sent to the application [8]. In order to speed up the process of sending and receiving files several cloud providers have offered a node-based solution which allows developers to connect to that endpoint using RESTful services and therefore circumventing the need to write code to specifically address each particular server or computer [8]. The architecture of these systems is simple and efficient but it lacks any checks or fallbacks when treating sensitive data. In other words, any type of file is just a simple file to these implementations, as seen below in Fig.1:

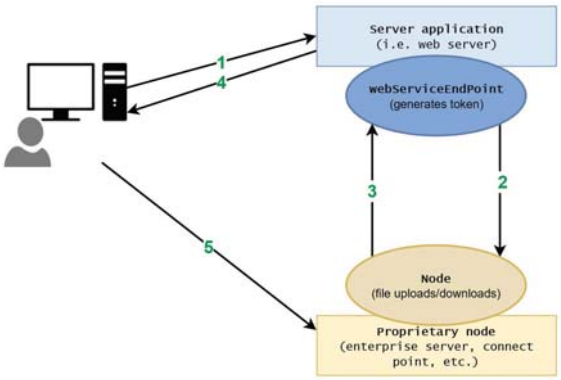

Fig. 1. The flow of a file transfer authorization token

Legend:

1. The initial request done to the server, this requires basic login information and specifics such as transfer paths and file names.
2. The JSON transfer request object is passed to a Node using REST API.
3. HTTP response containing information in JSON format.
4. Either the token by itself is returned or a confirmation code from the node that the file is available or if space has been created for the upload.
5. Using the returned token start the file transfer.

Despite the fact that the above workflow is seemingly secure none of the parts involved have any knowledge of what is transferred in between themselves. There are nearly no checks between the moment when the file is picked up from the node all the way back to the user's computer. This relies very heavily on the user or the privately-run cloud to implement a security policy that protects against malicious files and it too has to rely mostly on disseminating cybersecurity knowledge to all of its end users. Furthermore, if the files have sensitive content none of the parts involved are aware of this. This is not an issue in the strictest sense of the example illustrated in Figure 1, where we assume just 1 cloud connection used in a monolithic application, but when access to a node is given to multiple clients and services the dangers increase exponentially. In a micro-service-based architecture only one non-expired token or a slip-up in security administration can leave the entire repository of documents wide open to an attack.

Which is why we believe that a possible solution to this problem would be to feed the sensitive information and keywords right into the token and then use a context-aware deep learning algorithm to analyze the context of the files which are being requested or uploaded, as suggested by Yuya et. all. in their paper entitled "Context-Aware Data Loss Prevention for Cloud Storage Services". Simply modifying the token, instead of adding further security measures on the pipeline or in terms of file transfer, would be a quick, language-agnostic and backwards compatible solution that would be easy to deploy and develop for already existing platforms no matter how large or small [9].

```
public static String createFTT(String id, String issuer,
    List keywords, Long timeInEffect){

    SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;
    long timeInMilis = System.currentTimeMilis();
    Date now new Date(timeInMilis); Date exp;

    byte[] apiKey = DatatypeConverter.parseBase64Binary(SECRET_KEY);
    Key signingKey = new SecretKeySpec(apiKey,
        SignatureAlgorithm.getJcaName());

    FTTBuilder buider = FTTBuilder.builder().setID(id)
    .setIssuedAt(now)
    .setSubject(keywords)
    .setIssuer(issuer)
    .signWith(signature Algorithm, signingKey);

    if (timeInEffect >0){
        long expMilis = timeInMilis + timeInEffect;
        exp = new Date(expMilis);
        builder.setExpiration(exp);
    }else{
        long expMilis = timeInMilis + 3600000;
        exp = new Date(expMilis);
        builder.setExpiration(exp);
    }
    return builder.compact();
}
```

Fig. 2. Implementation of a file transfer token using Java

As can be seen in the above figure, an external JAVA library, namely *io.jsonwebtoken*, is used to implement the SignatureAlgorithm. The method also use the Java included algorithms for 256-bits. What it is necessary to specify is the fact that we used a specific class, FTTBuilder and we add information like a list of keywords to it. Also, because the token contains sensitive information we implicitly set an expiration time if no one was given.

When the requesting application needs to start a file transfer, instead of just sending an access token to the authorization server, it would send a combination of access token and ID token called file-transfer token. Inside the file-transfer token will be an encrypted string containing a set of key words or phrases which are either predetermined for that type of file or determined by the company's security policy, the type of file that is requested, the extension of the requested file or even by a deep-learning algorithm based on sensitive keywords that were part of files transferred by the application in the past [9]. The token will be unencrypted at the authorization server level and will be analyzed before the approval is sent back to the application. Decoding the token is something that should be done by each developer in particular using a decoding algorithm of their choice. However, we envision that the token should carry, like any other

commercially available tokens, a signature in its JSON message to indicate the origin of the message. In essence the token's actual contents will look as seen in table 1.

TABLE I. JSON TOKEN BREAKDOWN.

Header	{ "alg": "HS256", "typ": "JWT" }
Payload	{ "id": "1234567890", "time": "now()", "keywords": "patient, bank, technical, prototype", "issuer": "generateIssuerdId()" }
Signature	HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), signingKey())
Encoded result:	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjEyMzQ1Njc4OTAiLCJ0aW1lIjoibm93KCkiLCJrZXl3b3JkcyI6InBhdGllbnQsIGJhbmssIHRlY2hpbmNhbCwgcHJvdG90eXBlIiwiaXNzdWVyIjoiZ2VuZXJhdGVJc3N1ZXJkSWQoKSJ9.7gbYuAEHsT3lxHJeJti1DoWHfUSWIJhQWPcl83pg81M

We must mention that, obviously, the methods for time and *generatingIssuerIds* are just placeholders. Normally the method that generates these tokens will populate them with some information that the developers determines would make the token unique. Additionally, it must be mentioned that each of these fields will be sent through JSON and will not be separated as demonstrated above. Finally, the token will include meta-data (which is not presented above) on the time at which it was sent and when it was returned to the application, on expiration it will include the time it took the application to send it to the node and basic information on which node it was sent to, by which application, what file it returned or uploaded to which server and so on.

One major objection against this implementation would be that tokens containing a long list of key words would cause the request to become bloated and add unnecessary latency to responses, but we do not believe this to be the case. Given that ID tokens are commonly used already and contain quite a bit of information and might require the web server to interrogate several fields of a database in order to double check that information. The proposed solution does not involve the use of any database beyond the initial creation of the token or the document analysis system that will run after the token has already been confirmed as valid by the authorization server.

As far as the deep-learning mechanism is concerned, according to measurements done by Yuya Jeremy Ong et. all., the speed at which their experimental deep-learning algorithm, all of their implementations run at a below 100 ms speed even when doing a sentence by sentence keyword search [8][9]. This was achieved running on what is already considered outdated hardware, namely a single Nvidia 1080 Ti GPU and an i7-3820 CPU with 16 gigabytes of RAM. Even according to the latest consumer reports, the average load time of a page in eCommerce, one of the most performance hungry sectors of the industry, is between 0.9 and 1.1 seconds [10]. We can infer that only delays of above 150 ms would cause any loss of business due to slowness. Given that file transfer demands aren't nearly as instantaneous we are fully confident that even with the added delay companies and end users will not be impacted in any significant way.

However, the utility of this type of tokenization should not be reserved only for searching for key words that are considered sensitive. It could be used to specify that the receiving server will not accept obfuscated files, in order to prevent malicious scripts for being deployed client side. A study conducted in 2016 using multiple classifiers and machine learning algorithms on obfuscated JavaScript code have returned an accuracy of 99% in correctly classifying which files were obfuscated and which were not [11].

In this case the token could also work as a filter, informing the developer upon return that the file that they have requested contains obfuscated code, and, for example, transferring it is against the company's security policy. Much like in the case of specific keywords, analytics can be derived from the number of requests done per token-source for obfuscated content, which might help security auditors pinpoint a potential threat, bad practice or vulnerability in the security infrastructure.

In order to ensure analytics returned by the token dashboard or aggregation method are accurate even in vendor independent environments, such as multi-clouds, the originating point of the token should always be the physical machine where the application is initially run. This would require an implementation of the token that uses a logic similar to identity provider tokens [5], but with the added restrictions that only one point of contact, either an authentication server that works with multiple cloud providers or the final node, can renew the token or validate it. This would need to be done in order to ensure that no latency is added to each request as separate cloud providers use their version of the authenticator to check the token and send back a response to the issuing machine.

IV. MULTI-CLOUD TOKENIZATION

The main benefit of developing a file-transfer token system would come when using multiple cloud providers at once, as a token system dedicated to file transfers would allow not only better traceability of security threats but a better way of deploying containers, applications and micro-services. As a result, this section of the paper will focus on describing what we believe would be the problems that a file-transfer token solves as well as addressing some of the ways in which implementations can change or improve if adopting such a system.

Several challenges await security auditors and software developers who attempt to move their application to a vendor-independent cloud environment. Even at a basic, single cloud level, security is a tricky issue when deploying or running applications from servers that are not on premise. Cloud systems aren't aware of what is running in the cloud [12]. Multi-cloud systems, as a result, have an even greater lack of transparency when it comes to what part of the application is running in which cloud and what's coming in and going out. Furthermore, the entire issue of managing cybersecurity in cloud environments is hardly ever automated, relying greatly on measurements, analytics and human interpretation.

In the traditional approach to file transfer there is nothing to distinguish a file, be it malicious or of critical importance from just another text file of little importance. In order to actually improve multi-cloud security then, the file-transfer

tokenization security package will therefore have to answer multiple needs at one time:

TABLE II. SECURITY FEATURES OF FILE-TRANSFER TOKENS IN MULTI-CLOUD.

Title	Description
Token generation library	The library will contain provisions for generating the token, implementing analytics and reading the token.
Security overview	A dashboard like solution that will display the information on what file transfers are most common, failure points, deployment issues, etc.
Deployment tool	A DevOps tool that allows automated deployment and securing of packages when failure is detected in one part of the multi-cloud

The library that will be provided to developers will focus on a number of key components, other than the ones for generating the tokens. We've identified as a priority to offer developers the possibility of monitoring and gathering data on their issued tokens, number of issues and, especially contents of the issued tokens. In the last regard the library will contain multiple methods to interrogate the database created from the deep-learning algorithms or to be able to use a direct flow of information from the DL mechanism without having to rely on static data. In this sense a reactive approach to programming would be best, using Observables and Subscribers as ways of generating data flow and consuming it respectively.

In the event that the file has been broken up in multiple parts and sent to multiple cloud providers [13] the library will also have to provide methods to match each fragment of a file with another. This will be a key functionality of these tokens in multi-cloud environments as the meta-data that they contain on the file's origin and full size could help developers recompile the file and also determine a minimum bit loss [13] or corruption level at which the file would still be recoverable. Finally, the library will offer multiple ways of linking the data from the generated tokens, deep-learning algorithms and flows of data with the security overview dashboard. This means the implementation of methods to monitor, scan, enforce and manage incident notifications and other events.

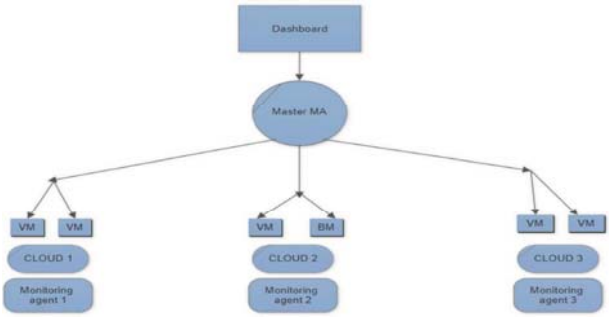

Fig. 3. Monitoring system architecture

We envision the security overview architecture to use a master monitoring agent (MA) and several individual monitoring agents for each cloud, placed just "behind" the cloud and the VMs so that information can be collected from multiple sources at once and not just at the moment when the file request is being sent and received from the VM or the cloud.

This monitoring solution will receive information from all of the other agents and will be informed in a timely fashion of issues happening in each individual cloud. If integrated with a security tool or a security framework such as MUSA [14], it will also be able to react to incidents and perform countermeasures to mitigate attacks or security risks. The data collected by the agents will be of several types, including file meta data, token data, keyword frequency, percentage of obfuscated files transferred and sensitivity of file contents that have been sent and processed.

The deployment tool will have 2 functional purposes for developers: it will enhance the security of containers and containerized data and it will be configurable to deploy automatically based on meta-data information or critical failures. Our choice of containerized deployment will be Docker, a very popular LxC which PAAS features. Docker works by creating a micro VM, a fully functional computer that contains the application developed, the configuration files, the library and system files and the OS kernel required for the application to run [15]. All of these elements are created as a single file that is then sent to the server.

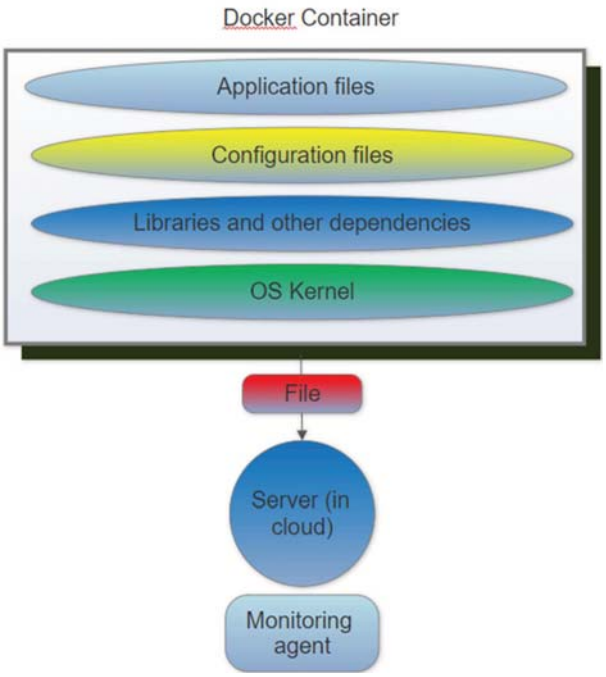

Fig. 4. Docker deployment workflow

While Docker has moved towards ensuring container security by isolating resources, including at a hardware level, application isolation, VM isolation and network isolation to name just a few [15], one issue still remains: the security outside of the container is still entirely left up to the deployment tool or the engineer coordinating the containers using tools such as Kubernetes. There are no de facto standards for even building secure containers, let alone for ensuring that the container reaches the server intact and that its sensitive data is not compromised. This is where our deployment tool will aim to change that by generating a file-transfer token upon containerization with the contents of the container, the sensitivity of those contents and any other relevant meta-data. When the receiving server will approve the incoming container all the monitoring agents will be

informed of what the contents of the new deployment are and of their criticality.

Finally, using the token meta-data developers can measure performance on any one cloud instance in which they are deploying or retrieving files from. By measuring delays between requests made and answer times a developer can create a load balancing tool to distribute files to a specific VM offered by a cloud provider that is not facing performance issues.

Additionally, this technique can be used to determine which clouds in use are the safest against file corruption or data loss when dispersing a file into several cloud systems and VMs. Ultimately, the purpose of all of this information that might seem redundant would be to ensure that data loss is kept to a minimum, and in that sense the information contained in tokens, especially that of the criticality and sensitivity of the files transferred makes it possible for developers to create fail-safe solutions far more easily.

One example of this could be that files with a certain level of criticality that are sent to a server, say credit card numbers or patients' private medical records, can be duplicated upon sending either in another cloud based VM or on a local server, as to prevent any data loss during transfer due to corruption or an unstable internet connection. Since the token will only initiate the file transfer after it's been approved by the server the risk of corrupting the file during transfer drops to zero.

V. Open Challenges

The challenges to implementing and making the file-transfer token system feasible go beyond those presented in earlier sections. The size of the token is only a small challenge facing any team that would want to implement this system.

The biggest challenge that the authors have identified is actually a combined issue: costs of implementation and the current cybersecurity landscape. We chose to fuse both of these seemingly unrelated issues because, upon analysis, we found that they are very much related. Starting with the cost of implementation, we do not believe it to be very high in terms of generating, authenticating and using the tokens. Since the file-transfer token is based on a common OAuth 2.0 standard changing its contents and adding a few strings of text is not a complicated project. Neither is reading those few strings.

Where complications arise though is when implementing the deep-learning algorithms required to identify the criticality and sensitivity of data. Firstly, these systems have only been developed at a research level and as a result there are no commercially available solutions for companies to just buy and setup/deploy [9]. Secondly, the technical knowledge, not only in terms of software and AI development but in terms of morphological analysis required to implement a deep-learning mechanism are fairly high. Finally, a small but considerable challenge would come from creating server farms sufficiency powerful to run deep-learning systems that are efficient in identifying thousands of key words from hundreds of thousands of documents in a timely fashion.

These concerns above go hand in hand with most companies' policy towards cost saving which also affects the cybersecurity landscape today. Most cybersecurity today is not automated and decisions are made by CSOs or other security personnel based on analysis, international standards or company policy [16]. The proposed solution is expensive because it means creating and maintaining a huge framework for token data collection, a dashboard to display the information and a security policy that restricts deployment of containers in certain locations based on the criticality of the application or other criteria. It's a combination of both opportunity cost and real costs. In terms of opportunity costs, we've identified a loss in speed with deployment, less flexibility of hardware used (both physical and in the cloud) and further analysis time required to identify sensitive keywords or troublesome nodes in the file distribution system. Real costs, beyond the obvious costs of development, would also be incurred with the training of employees.

Developers handling sensitive files would need to know what the company considers sensitive data, to know how to deal with it in the context of the new tokenization and where they can place it. Other employees would also need to be made aware of the new security restrictions.

The second biggest challenge we've identified is related to the standard already in place, which might be hard to change. Securing the token and file is currently a hand-in-hand operation for most software developers. They can secure the request pipeline by using SSL and then send the file via that same secured pipeline once the token has been authorized.

The entire system envisioned above can also be replicated using just simple tokens which might be more convenient for some. In order for file-transfer tokenization to work it would need to be adopted en masse, because only then could providers offer deep-learning libraries which could be targeted to specific sectors and could be taught to understand how to distinguish between malicious files and just regular data with some level of encryption. Failing to have many developers adhere to a new standard would also complicate access to files which are using one of the other types of tokens at either end. This would make it difficult for companies that want to invest in multi-cloud solutions to invest in those cloud providers that do or do not offer the tokenization system that they prefer.

The way that the software industry is currently developing we have no doubt that the need for innovative tokenization systems is going to increase, but currently the use of multiple cloud vendors at once, which is the key to gaining major benefits from using a file-transfer token system, is sparse and is being blocked by challenges which are not always technical in nature such as harmonizing SLAs, measuring costs or performance [17]. Currently there simply isn't a standardized methodology to measure performance in multi-cloud systems which in turn severely impedes the quicker adoption of multi-clouds and limits the amount of information about cloud performance itself [18].

Finally, much of what was discussed in this paper is a theoretical approach to solving security challenges with file transfers. As much as possible, we, the authors of this paper, have attempted to only use resources and ideas where experiments have already been conducted successfully or commercially available tools.

That being said, there is still a large gap between a controlled environment with top of the line hardware and knowledgeable engineers running a clear-cut, precisely designed deep-learning AI and the real-world equivalent of said AI. The gap that software development finds more and more difficult to bridge isn't necessarily that of implementation but design and especially design by committee or customer requirements.

At a purely implementation level there is no doubt that solutions can cross over from the theoretical sphere to the commercial but it's when they hit unrealistic deadlines with impossible demands and using very limited resources that these solutions begin to falter.

In order for this transition to be successful the teams implementing it would need to be focused as much as possible on the end goal, would probably need to adopt a Waterfall-like project management system with clearly outlined steps and exit criteria and would need to abstain from adding features that would not be necessary as to not bloat the costs of development or maintenance. It's important to keep in mind that the implementation costs of these tokens are entirely supported by the software developer and they most likely not benefit the end user directly.

VI. Conclusions

When analyzing security challenges, one must keep in mind that cyber security doesn't necessarily mean just a programmatic level fix, better code or applications in isolation. The challenge in keeping today's computers, networks and applications secure is complex and involves many moving parts including a very large human factor in the form of end users.

This is why the authors of this book believe that any level of automation and any reduction in the human factor involved in securing computers is key for the future development of applications in shared computing spaces such as the cloud. As applications become increasingly less monolithic and more geared towards microservices and dynamic deployment a level of security independence needs to be reached by each application in part, lest the effort of preventing attacks becomes too much to realistically be bared by security officers. We believe our approach is one of a number of possible solutions in helping solve these issues.

What file-transfer tokens hope to achieve is to give more respite to IT security specialists, forewarn them of any possible attacks, pinpoint a potential weak spot and delegate as much of the task of actively protecting the system to the system itself and not to the end users. These tokens also help developers secure their own work better and ensure that the security process is started from the lowest level of the design and development level possible.

Reducing the time to detect failure or security issues reduces the cost of fixing them since these can be discovered in code, should it not be written to use this type of token. But even going beyond the code the security threat can be pinpointed at several different intervals and, more importantly, can be quantified. By adding a machine learning component to the tokenization system, we give developers and cybersecurity specialists a way to identify the most sensitive data being sent and where the end points of these operations are. At that time, they can take the measures they deem necessary to ensure optimal data protection and prevent data loss which, as surmised, is the most critical of all security problems encountered by IT companies worldwide.

Acknowledgment

The research presented in this paper was supported by the following projects: ROBIN (PN-III-P1-1.2-PCCDI-2017-0734) and SeMed (2933/55/GNaC 2018).

References

[1] R. Velumadhava Rao, K. Selvamani, "Data Security Challenges and Its Solutions in Cloud" Computing. Procedia Computer Science. 48. pp 204-209, 2015

[2] Auth0 Documentations, "Tokens used by Auth0", 2019, https://auth0.com/docs/tokens,
[Online: accessed February 2019]

[3] Auth0 Documentations, "Access Tokens", 2019, https://auth0.com/docs/tokens/overview-access-tokens
[Online: accessed February 2019]

[4] Auth0 Documentations, "ID Token", 2019, https://auth0.com/docs/tokens/id-token
[Online: accessed February 2019]

[5] Auth0 Documentations, "Identity Provider Access Tokens", 2019, https://auth0.com/docs/tokens/overview-idp-access-tokens
[Online: accessed February 2019]

[6] Auth0 Documentations, "Refresh Token", 2019, https://auth0.com/docs/tokens/refresh-token/current
[Online: accessed February 2019]

[7] Prajakta Solapurkar, "Building secure healthcare services using OAuth 2.0 and JSON web token in IOT cloud scenario", 2nd International Conference on Contemporary Computing and Informatics, Noida, India, 14-17 Dec. 2016

[8] IBM Aspera Developer Documentation, "Node API Authentication and Authorization", 2019,
https://developer.asperasoft.com/web/node/auth
[Online: accessed February 2019].

[9] Yuya Jeremy Ong, Mu Qiao, Ramani Routray, Roger Raphael, "Context-Aware Data Loss Prevention for Cloud Storage Services", 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), 2017

[10] Akamai, "State of Online Retail Performance: 2017 Holiday Retrospective", pp 7, 2018

[11] Bernhard Tellenbach, Sergio Paganoni ,Marc Rennhard, "Detecting obfuscated JavaScripts from known and unknown obfuscators using machine learning", International Journal on Advances in Security, pp 196 – 206, 2016.

[12] Antonio M. Ortiz, Erkuden Rios, Wissam Mallouli, Eider Iturbe and Edgardo Montes de Oca; "Self-protecting multi-cloud applications", 2015 IEEE Conference on Communications and Network Security (CNS), pp 643, 2015.

[13] Bharat B. Madan, Manoj Banik, Bo Chen Wu, Doina Bein, "Intrusion Tolerant Multi-Cloud Distributed Storage", 2016 IEEE International Conference on Smart Cloud (SmartCloud), pp 262 – 268, 2016.

[14] MUSA: Multi-Cloud Secure Applciation, a Framework for Faculitating Security in Multi-Cloud Applications,
http://www.tut.fi/musa-project/, accessed on January 2019

[15] A R Manu, Jitendra Kumar Patel, Shakil Akhtar, V K Agrawal, K N Bala Subramanya Murthy; "Docker container security via heuristics-based multilateral security-conceptual and pragmatic study", 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT), 2016

[16] Hassan Takabi, James B.D. Joshi and Gail-Joon Ahn – "Security and Privacy Challenges in Cloud Computing Environments", IEEE Security & Privacy, 8(6), pp 24-31.

[17] Vincent C. Emeakaroha, Marco A.S. Netto, Rodrigo N. Calheiros, Ivona Brandic, Rajkumar Buyya, César A.F. De Rose, "Towards autonomic detection of SLA violations in Cloud infrastructures", Future Generation Computer Systems, 28(7), pp 1020, 2012.

[18] D. Kossmann, T. Kraska and S. Loesing, „An Evaluation of Alternative Architectures for Transaction Processing in the Cloud", Proceedings of the 2010 ACM, pp. 1-12, 2010