Serverless Computing Security: Protecting Application Logic

Conference Paper · June 2020

DOI: 10.1109/ISSC49989.2020.9180214

Ruth G. Lennon
Atlantic Technoloical University (ATU) Donegal
36 PUBLICATIONS **158** CITATIONS

Serverless Computing Security: Protecting Application Logic

Wesley O'Meara
Dept. of Computing
Letterkenny Institute of Technology
Letterkenny, Co. Donegal, Ireland
womeara@gmail.com

Ruth G. Lennon
Dept. of Computing
Letterkenny Institute of Technology
Letterkenny, Co. Donegal, Ireland
Ruth.Lennon@lyit.ie

Abstract— Serverless computing enables organisations to avail of the inherent and unlimited flexibility and scalability that serverless provides, without having to consider the underlying infrastructure. However, there are security considerations that are unique to serverless architectures, that if not included early in application design, can lead to vulnerabilities which could be exposed to common attack vectors. While cloud service providers manage the security of the underlying infrastructure, it is up to the consumer to ensure that serverless applications are fully protected. We go on to discuss common attack vectors, the risks associated with misconfiguration within security and application setup, how attackers target vulnerabilities within the workflow logic of serverless applications and their functions to focus their attacks, and how consumers can implement measures to protect their applications within a serverless architecture.

Keywords—serverless, function as a service, cybersecurity, application security, cloud computing.

I. Introduction

As serverless computing becomes increasingly prevalent across the industry, expenditure on Serverless technologies is projected to have an annual growth rate of 33% [1], increasing from $1.9 billion in 2016, to a projected $7.7 billion by 2021. IT departments are examining this new service offering with a view to enabling their business partners gain value from adopting serverless and transitioning their infrastructure and services to the cloud.

As a cloud computing service model, serverless offers consumers the ability to build and host event-driven applications on pooled resources [2]. The workflow of these applications consists of a series of (functions), that execute upon predefined events (triggers) [3]. All of which is supported by a serverless service model that will dynamically allocate resources in response to demand. The benefits for consumers, from building out their applications with the serverless model, is that there is infinite elasticity with the underlying infrastructure, minimal costs [4] within the pay-as-you-go model for the service usage, easy to use interface and configuration [5], and no requirement for consumers to understand or visualize the underlying infrastructure.

However, as consumers adopt serverless, issues can arise from the speed and complexity of this transition, coupled with the need to adhere to statutory and security requirements. This can be challenging for organisations that may want quickly seize business opportunities but may be short of technical expertise and resources. Due to microservices architecture of serverless applications, development and support teams can incur additional complexity as the design of serverless applications include creating application workflows by combining multiple cloud services, functions, triggers and events. Configuration and security administration of these components is implemented at the individual component level and as the architecture is event driven, each component can be triggered from a range of sources [6] such as e.g. user input, database updates or storage events. Another factor is that security scanning and testing tools, used within traditional software development, have not been able to adapt [7] to the stateless architectural nature of serverless. Consumers need to ensure that they implement a strong and thorough security strategy for the application design and security implementation or potentially they could leave their serverless architecture vulnerable to cyber-attack.

Cybercriminals target the applications that consumers host on cloud services, with the intention of profiting from illegally accessing sensitive data, denying services to the consumer or utilizing the cloud resources for their own purposes. One aspect that affects their ability to gain unauthorized access to a consumers' cloud resources, is whether they can visualize and map the internal business logic and application workflow, by gaining access to execution timings and patterns of functions and other cloud services, that drives a consumers serverless hosted application. Gaining an understanding of an applications workflow can enable an attacker to focus on weaknesses, such as vulnerabilities within the flow of functions [7], within the application architecture and utilize multiple attack vectors to try circumvent security measures or input validation processes, and allow them to attack critical services directly.

To fully realise the benefits of serverless while protecting the integrity of their applications and sensitive data, consumers need to consider these types of attacks when designing and implementing their security strategies and application architectures, and gain an understanding of what aspects of the architecture are vulnerable to malicious attack.

II. Background

Serverless, known as function-as-a-service (FaaS) [8] and backend-as-a-service (BaaS), is a new service model being offered for public consumption, by cloud service providers (CSP). Amongst the CSPs, Amazon was the first to launch a public serverless service, called Lambda, back in 2014 [9], other CSPs quickly followed with Google, Microsoft and IBM all devising and offering their serverless services to the public. Serverless enables cloud consumers to build and host applications, whose workflows that are triggered by events [10], where billing for service usage is granular, and the application owners and development do not have to concern themselves with the supporting infrastructure.

Usage of the term, serverless, isn't accurate as it still relies on servers within the cloud infrastructure, but the term does enable consumers to better visualise their role within its usage, as they are not required to have visibility of, or configure, any of the underlying infrastructure [11], i.e. Servers.

Serverless services enable consumers to create and host applications that are designed within microservices architectures, on CSP provided cloud infrastructure. The computing logic within these applications is provided by event driven action-based units of code called functions [12]., each with a single purpose. These functions can then be connected together with additional cloud services to create business workflows. The use cases for serverless include supporting event based applications, such as multimedia processing or the Internet of Things (IoT). For multimedia processing, e.g. a service can be provided to the public that enables them to upload a file to an S3 bucket. Once that file is uploaded, it will trigger a downstream function that will execute transformational processes upon that file and return it to the user. Another use case would be to support the Internet of Things (IoT), where it can enable the ability to respond to sensor input messages and automatically scale in response to the volume of input received.

Application workflows within serverless follow an event driven model, as the functions are invoked by an event, called a *trigger* [13]. These triggers can be initiated by actions such as e.g. user input, or changes within a database, and as these functions are completely stateless, they can be scaled independently [14]. Serverless is best suited to applications that are stateless, event-driven and short running [15]. To ensure that consumers design their functions to be small independent units of code, CSPs enforce restrictions on the size and runtime of the functions execution. Typically, this is up to 256MB for the code and a runtime max 5 minutes [16] for the function.

One of the key advantages that serverless provides to consumers is that they no longer need to consider the underlying infrastructure as part of the application design. Allowing development team to rapidly develop code for applications without needing to consider the underlying infrastructure [17]. This is due to the CSP taking responsibility for the cloud infrastructure, managing security, performance, patching etc., while also ensuring that the infrastructure elastically scales appropriately [18] during periods of increased demand.

Another benefit of adopting serverless is that potential to reduce costs for the consumer, CSPs offer a pay-as-you-go model for usage of their services and only charge at a granular level for the services that consumers actually use e.g. costs are incurred solely for the timeframe that a service, e.g. a function, is in use, [16]. As the CSPS are hosting and managing the cloud infrastructure, consumers no longer need their own data centres and can avoid the financial investment in static infrastructure.

III. SECURITY CONSIDERATIONS

The microservices, event-driven nature of a Serverless architecture can increase the attack surface of an application and can make it vulnerable to malicious attacks.

In 2017, OWASP released a report [6] that detailed the most common attack and risks associated with serverless, these were*:*

1. *Code Injection* – can potentially allow attackers to alter the execution pattern of code by injecting malicious code into the application.
2. *Broken Authentication* - Enables access to services by the capture or bypass of authentication methods.
3. *Sensitive Data Exposure* – Data breaches where unauthorised access to obtained to sensitive data.
4. *XML External Entities (XXE)* - Similar to a code injection attack that exploits any application that parses XML input.
5. *Broken Access Control* – can potentially allow users to perform tasks that may not be within their role or responsibilities.
6. *Security Misconfiguration*s – if the design and implementation of security protocols is not thorough, attackers may be able to exploit these to gain unauthorised access to services.
7. *Cross-Site Scripting (XSS)* – cybercriminals could potentially modify the code that the frontend of an application delivers to the users' browser.
8. *Insecure Deserialization* - where untrusted or unknown data is injected into an application to try affect the application logic and e.g. attempt to bypass authentication or affect the stability of the application.
9. *Using Components with Known Vulnerabilities* – Updates to tools are often released, that include patches to fix vulnerabilities that may have been discovered after the tool was released. Cyber criminals are also aware of these published vulnerabilities and may utilise these to focus their attacks.
10. *Insufficient Logging and Monitoring* – enables cyber-attacks to go unnoticed as consumers are unable to identify that their applications have been compromised or their services are used for illicit purposes.

Another such attack vector is for cybercriminals to target vulnerabilities produced by the application of inadequate security protections, and authentication protocols to functions, thereby weakening the integrity of the application flow between these functions [19]. These "*Business logic/Flow Manipulation*" [6] or "*Function-Flow Vulnerabilities"* [7] can enable attackers to identify where, within an application workflow, they can attempt to bypass the application logic and manipulate its workflow. Potentially enabling attackers to pass unverified data or commands directly into downstream functions for processing, allowing them to bypass access controls, change user access privileges or disrupt the applications stability.

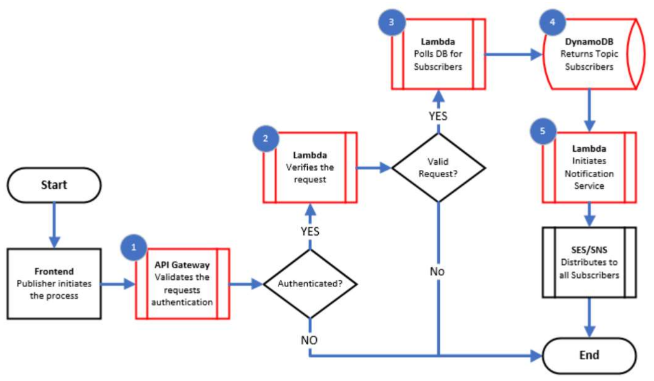

Start
Frontend
Publisher initiates the process
1
API Gateway
Validates the requests authentication
Authenticated?
NO
YES
2
Lambda
Verifies the request
Valid Request?
YES
No
3
Lambda
Polls DB for Subscribers
4
DynamoDB
Returns Topic Subscribers
5
Lambda
Initiates Notification Service
SES/SNS
Distributes to all Subscribers
End

Figure 1 Attack points for workflow manipulation

Figure 1, illustrates the possible attack points within a sample serverless application, illustrating a Publication component of a Pub/Sub (Publication/Subscription) use case, workflow where vulnerabilities could be manipulated to divert the workflow:

1. *API Gateway* is used to validate the authentication of requests. Vulnerabilities in the authentication protocols could enable the frontend to be bypassed via direct utilisation of APIs to send input to the API Gateway.
2. *Lambda* is used to verify if requests and their content are valid. Vulnerabilities can exist as functions can consume malicious input from unexpected sources across multiple function triggers.
3. *Lambda* is also used process the request, this can introduce vulnerabilities if authentication protocols are weak and direct utilisation of APIs is used to send input directly to this function, bypassing all verification steps
4. *DynamoDB* is used for record creation in this scenario. Gaps in database security could enable data breaches as unauthorised commands could expose sensitive data.
5. *Simple Email System (SES)* is used to distribute information via email. If poorly protected, this can be a means for attackers to send sensitive data outside of the application architecture

Figure 2, illustrates how a function workflow could be manipulated to directly access the DyanmoDB instance, bypassing all authentication and validation checks, and potentially exposing sensitive data

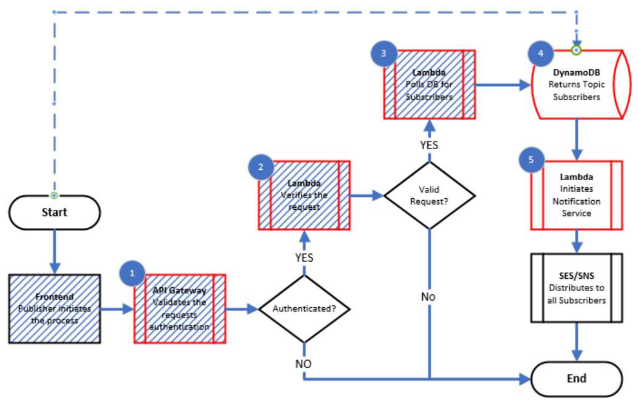

Start
Frontend
Publisher initiates the process
1
API Gateway
Validates the requests authentication
Authenticated?
NO
YES
2
Lambda
Verifies the request
Valid Request?
YES
No
3
Lambda
Polls DB for Subscribers
4
DynamoDB
Returns Topic Subscribers
5
Lambda
Initiates Notification Service
SES/SNS
Distributes to all Subscribers
End

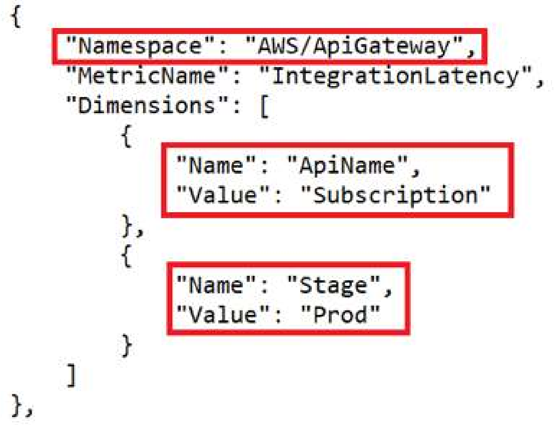
Figure 2 Bypassing verifications steps via workflow manipulation

IV. Methodology

As outlined by OWASP [6], an attacker can exploit security misconfiguration such as *Broken Authentication* or *Broken Access Control*, to enable the attacker to remotely execute code or access standard AWS commands, or sdks, to gather pertinent information on the application workflow from Amazons application and infrastructure monitoring service, CloudWatch.

A simple *list-metrics* command, executed against CloudWatch, will output the information sampled in Figure 3, 4 and 5.

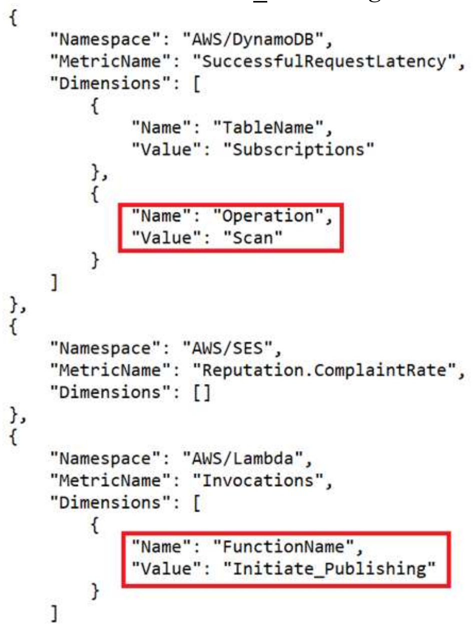
The output displayed in Figure 3 identifies an API Gateway resource called *Subscription* which is configured as a Production stage.



```
{
    "Namespace": "AWS/ApiGateway",
    "MetricName": "IntegrationLatency",
    "Dimensions": [
        {
            "Name": "ApiName",
            "Value": "Subscription"
        },
        {
            "Name": "Stage",
            "Value": "Prod"
        }
    ]
},
```

Figure 3 Identifies an API Gateway resource called Subscription

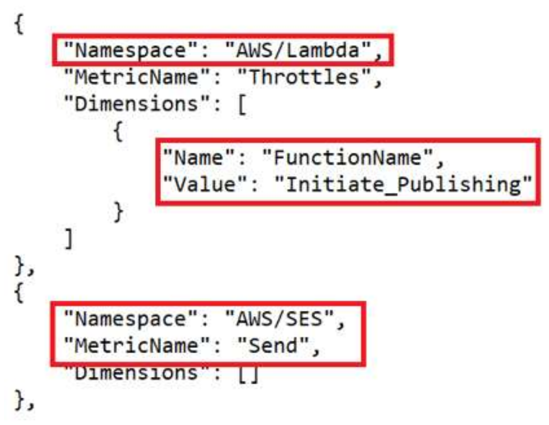
Figure 4 identifies a DynamoDB instance with a table called *Subscriptions*, and SCAN operation against that Table by a Lambda function called *Initiate_Publishing*.



```
{
    "Namespace": "AWS/DynamoDB",
    "MetricName": "SuccessfulRequestLatency",
    "Dimensions": [
        {
            "Name": "TableName",
            "Value": "Subscriptions"
        },
        {
            "Name": "Operation",
            "Value": "Scan"
        }
    ]
},
{
    "Namespace": "AWS/SES",
    "MetricName": "Reputation.ComplaintRate",
    "Dimensions": []
},
{
    "Namespace": "AWS/Lambda",
    "MetricName": "Invocations",
    "Dimensions": [
        {
            "Name": "FunctionName",
            "Value": "Initiate_Publishing"
        }
    ]
```

Figure 4 Identifies a functions capability within DynamoDB

Figure 5 details that the Lambda function is triggering a SEND event within SES.



```
{
    "Namespace": "AWS/Lambda",
    "MetricName": "Throttles",
    "Dimensions": [
        {
            "Name": "FunctionName",
            "Value": "Initiate_Publishing"
        }
    ]
},
{
    "Namespace": "AWS/SES",
    "MetricName": "Send",
    "Dimensions": []
},
```

Figure 5 Lambda Function triggering a SES SEND event.

V. RESULTS

Utilising the information gained in Figures 3, 4 and 5, an attacker can chain together the connected services and begin to infer flow within the application logic. Further probing would be necessary to fully determine the input requirements of the functions but it is enough information to infer that a API called *Subscription* is triggering a Lambda function called *Initiate_Publishing,* which is executing a SCAN operation against a DynamoDB table called *Subscription*s. It also details that the function has the capability to call AWSs Simple Email System (SES). Figure 6 illustrates the points within the workflow which an attacker was able to identify via the CloudWatch logs.

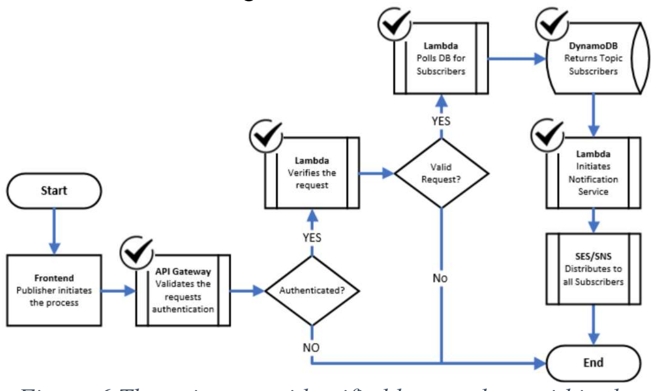

Start
Frontend Publisher initiates the process
API Gateway Validates the requests authentication
Authenticated?
YES
NO
Lambda Verifies the request
Valid Request?
YES
No
Lambda Polls DB for Subscribers
DynamoDB Returns Topic Subscribers
Lambda Initiates Notification Service
SES/SNS Distributes to all Subscribers
End

Figure 6 The points now identified by attackers within the application workflow

As a possible attack vectors, it is conceivable that an attacker could utilise this information and focus their attack on the SCAN functionality, within the function, to try read data from the database and export that data via the SES connection to that function. Resulting in a data breach for the legitimate consumer which could lead to financial losses or reputational damage.

VI. SECURING SERVERLESS APPLICATIONS

Amazons' AWS documentation separates cloud security ownership into two realms: (1) the Cloud Service Providers responsibility for all security of the underlying cloud infrastructure, which Amazon term as *security "of" the cloud*; (2) and the consumers' responsibility for ensuring that the security of their applications hosted on AWS essentially *security "in" the cloud* [20]. From a consumers' perspective, this means that they are responsible for ensuring that serverless applications design follow best practices. Amazon will ensure that the underlying infrastructure is secure and that AWS provides the functionality to enact security policies at a granular level. If the consumer follows best practice, they should be able to prevent the common serverless attack vectors that OWASP list in their Serverless Top 10.

As the majority of attack vectors focus their efforts on exploiting weaknesses within a consumers' configuration [21] of serverless services, this is where the most value can be gained from ensuring that application architectures are protected from these type of vulnerabilities. Implementing automation and designing architecture with security in mind can enable cloud consumers to develop applications where security is inherent in its design.

There are a number of areas across resourcing, architecture, automation and monitoring where consumers can strengthen their security mindset, these are:

- *Education the decision makers and support roles* - ensure that all decision makers and operations support roles have the adequate skills and training required to implement best practice across all areas of the serverless architecture, especially within network, database and application security.
- *Introduce a culture of collaboration* - As the line between the traditional roles within the IT organisations is diminished, implementing a DevOps culture has become even more important. A culture of collaboration, with a focus on DevSecOps principles for security across all stakeholders responsible for the application architecture, will foster knowledge sharing across all teams and enable the business to holistically introduce a security mind-set with the aim of reducing security vulnerabilities.
- *Design architecture with security in mind* – ensure that security is at the forefront of all architecture design considerations. For example, designing the application architecture to follow a three-tier (or multi-tier) architecture consisting of a presentation, application and data layer. Clearly defining these layers enables a clearer focus on the interfaces between these layers and allows for security realms to be tailored for each layer. Understanding the interfaces and interactions between the layers would enable the expected behaviour of an application be clearer and aide the highlighting of unexpected or malicious activities. This would have multiple benefits such as enabling the system to be more robust, scale independently and be more resilient to attack.
- *Introduce Threat Modelling* – use threat modelling as a means of systematically analysing the controls and methods of defence needed to neutralise attack vectors used by cybercriminals.
- *Introduce limitation of authorisation* – segregate the security realms between the layers of the architecture and

ensure that user and service accounts are only authorised to access the services that are required for their roles.

- *Implement secure coding standards* – secure coding can aide the prevention of security vulnerabilities within the application code itself. Such practices can cover areas such as input validation, output encoding, session management, error handling and logging.
- *Automate and secure deployment systems* – remove manual intervention by automating deployments where possible, implementing infrastructure as code and release artefact management for all infrastructure setup and configuration. The introduction of DevOps and its utilisation of CI/CD practices, i.e. Continuous Integration/Continuous Delivery and Continuous Deployment, combined with a DevSecOps focus on Security, can ensure that deployment processes are built with security in mind. Enforce governance of deployment infrastructure via segregation of duties, audits and security checks.
- *Continuous Monitoring* - Ensure that consumers have visibility of all cloud services and usage. Implement monitoring and metrics packages, create processes for the handling of incidents, continually scan for vulnerabilities within the architecture, categorise risk.

VII. CONCLUSIONS

This research has tried to analyse common attack patterns within serverless and make recommendations on counteracting these types of attacks. As the majority of attack patterns are reliant on misconfiguration of serverless services [21], it is imperative that consumers incorporate security considerations early in their application lifecycle and put practices in place to continually implement and enforce these practices throughout the lifespan of their applications.

We have included a discussion on serverless security vulnerabilities associated with the exploitation of function flow vulnerabilities, especially as the services and capabilities provided by cloud service providers mature, so do the attack vectors utilized by cybercriminals. This ensures that security issues with serverless architectures remain relevant and should be an ongoing consideration for all serverless consumers.

As long as these continue, further research is needed to counteract these attack vectors and ensure that consumers are protected.

ACKNOWLEDGMENT

The authors would like to thank Letterkenny Institute of Technology for their funding of this research work.

REFERENCES

[1] Taylor, H., 2019. 2020 CYBERSECURITY PREDICTIONS FOR SOFTWARE DEVELOPMENT AND ENTERPRISE ARCHITECTURE. ttps://journalofcyberpolicy.com/2019/12/21/2020-cybersecurity-predictions-software-development-enterprise-architecture/

[2] Lynn, T., Rosati, P., Lejeune, A., Emeakaroha, V., 2017. A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE. p162.

[3] Glikson, A., Nastić, S., Dustdar, S., 2017. Deviceless edge computing: extending serverless computing to the edge of the network.Publication: SYSTOR '17: Proceedings of the 10th ACM International Systems and Storage Conference. ACM. p1

[4] Kritikos, K., Skrzypek, P., 2018. A Review of Serverless Frameworks. 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). IEEE. p161.

[5] Kim, J., Park, J., Lee, K., 2019. Network Resource Isolation in Serverless Cloud Function Service. 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W). IEEE. p183.

[6] Melamed, T., 2017. OWASP Top 10 (2017) Interpretation for Serverless. https://www.owasp.org/images/5/5c/OWASP-Top-10-Serverless-Interpretation-en.pdf.

[7] Patnayakuni, R., Patnayakuni, N., 2018. Securing Serverless Computing. (SIGSEC). Association for Information Systems AIS Electronic WISP 2018 Proceedings Pre-ICIS Workshop on Information Security and Privacy Library (AISeL).https://aisel.aisnet.org/wisp2018/15/.

[8] Kanso, A., Youssef, A., 2017. Serverless: beyond the cloud. WoSC '17: Proceedings of the 2nd International Workshop on Serverless Computing. ACM. p6.

[9] Asghar, T., Rasool, S., Iqbal, M., ul Qayyum, Z., Noor Mian, A., Ubakanma, G. Feasibility of Serverless Cloud Services for Disaster Management Information Systems. 2018 IEEE 20th International Conference on High Performance Computing and Communications. IEEE

[10] van Eyk, E., Toader, L., Talluri, S., Versluis, L., Uță, A., Iosup, A. (2018). Serverless is More: From PaaS to Present Cloud Computing. Published in: IEEE Internet Computing (Volume: 22, Issue: 5, Sep./Oct. 2018). IEEE. p9.

[11] Lee, H., Satyam, K., Fox, G., 2018. Evaluation of Production Serverless Computing Environments. In 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). IEEE. p442.

[12] Saha, A., Jindal, S., (2018). EMARS: Efficient Management and Allocation of Resources in Serverless. Published in: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). IEEE. p827

[13] Parres-Peredo, A., Piza-Davila, I., Cervantes, F., 2019. Building and Evaluating User Network Profiles for Cybersecurity Using Serverless Architecture. 2019 42nd International Conference on Telecommunications and Signal Processing (TSP). IEEE. p165.

[14] Mohanty, S., Premsankar, G., di Francesco, M., 2018. An Evaluation of Open Source Serverless Computing Frameworks. 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE. p115.

[15] Feng, L., Kudva, P., Da Silva, D., Hu, J., 2018. Exploring Serverless Computing for Neural Network Training. 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). IEEE. p334.

[16] Lloyd, W., Ramesh, S., Chinthalapati, S., Ly, L., Pallickara, S., 2018. Serverless Computing: An Investigation of Factors Influencing Microservice Performance. 2018 IEEE International Conference on Cloud Engineering (IC2E). IEEE. p159.

[17] Sewak, M., Singh, S., 2018. Winning in the Era of Serverless Computing and Function as a Service. 2018 3rd International Conference for Convergence in Technology (I2CT). IEEE. p1

[18] Adzic, G., Chatley, R., 2017. Serverless computing: economic and architectural impact. ESEC/FSE 2017: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM.

[19] Thomas, I., 2018. Top 10 Security Risks In Serverless. https://www.we45.com/blog/top-10-security-risks-in-serverless.

[20] Amazon, 2020. Security in AWS Security Hub. https://docs.aws.amazon.com/securityhub/latest/userguide/security.html

[21] Radichel, T., 2020. Serverless Attack Vectors. RSA Conference 2020.