

Securing Weak Points in Serverless Architectures

Risks and Recommendations

Alfredo de Oliveira

Securing Weak Points in Serverless Architectures: Risks and Recommendations

Published by
Trend Micro Research

Written by
Alfredo de Oliveira
Senior Threat Researcher

Stock image used under license from
Shutterstock.com

TREND MICRO LEGAL DISCLAIMER

The information provided herein is for general information and educational purposes only. It is not intended and should not be construed to constitute legal advice. The information contained herein may not be applicable to all situations and may not reflect the most current situation. Nothing contained herein should be relied on or acted upon without the benefit of legal advice based on the particular facts and circumstances presented and nothing herein should be construed otherwise. Trend Micro reserves the right to modify the contents of this document at any time without prior notice.

Translations of any material into other languages are intended solely as a convenience. Translation accuracy is not guaranteed nor implied. If any questions arise related to the accuracy of a translation, please refer to the original language official version of the document. Any discrepancies or differences created in the translation are not binding and have no legal effect for compliance or enforcement purposes.

Although Trend Micro uses reasonable efforts to include accurate and up-to-date information herein, Trend Micro makes no warranties or representations of any kind as to its accuracy, currency, or completeness. You agree that access to and use of and reliance on this document and the content thereof is at your own risk. Trend Micro disclaims all warranties of any kind, express or implied. Neither Trend Micro nor any party involved in creating, producing, or delivering this document shall be liable for any consequence, loss, or damage, including direct, indirect, special, consequential, loss of business profits, or special damages, whatsoever arising out of access to, use of, or inability to use, or in connection with the use of this document, or any errors or omissions in the content thereof. Use of this information constitutes acceptance for use in an "as is" condition.

Securing Weak Points in Serverless Architectures: Risks and Recommendations

Contents	Serverless Architectures	5
	Connected Services in a Serverless Architecture	7
	Misconfigurations and Unsecure Coding Practices	10
	Possible Compromise and Attack Scenarios	18
	Other Security Considerations in Serverless Deployments	20
	Security Measures for Serverless Services	23
	Serverless Technology and Shared Responsibility	27

The public cloud has empowered enterprises to reach new digital heights, allowing them to create dynamic and scalable operations. For their varying dynamic and flexible needs, there are different compute options available for enterprises to choose from.¹ One of those is the serverless model.

Serverless computing is a kind of cloud computing execution model that enables enterprises to use the computational power of a cloud service provider (CSP), such as Amazon Web Services (AWS). It allows enterprises to take advantage of a further reduction in overhead expenses pertaining to server operations and maintenance and to associated processes such as patch management, scaling, and availability. With serverless computing, enterprises can focus on building apps and core products, rather than using manpower to maintain and secure server infrastructure. This means that enterprises that choose to go serverless benefit from increased flexibility, automation, cost-effectiveness, and agility.

From powering and scaling websites and applications in a matter of minutes² without requiring adopters to worry about infrastructure, to allowing organizations to iterate software faster using the continuous integration and continuous deployment (CI/CD) methodology, serverless technology is enabling organizations to have the speed and the efficiency that they need to drive innovation and improve business.

The serverless model is regarded as relatively more secure than other cloud models because, for example, in the case of AWS Lambda, AWS takes care of the underlying infrastructure, the operating system, and the application platform. But this does not mean that securing the serverless model falls solely under AWS' responsibility. AWS Lambda users are responsible for securing their code, the storage and the accessibility of sensitive data, and the identity and access management (IAM) in relation to the AWS Lambda service and within their function. In short, the services that users choose to use dictate what they are responsible for. The serverless model also requires customers to understand their responsibility in maintaining proper IAM, critical data storage and accessibility, and code quality. CloudOps and DevOps professionals need to be responsible in properly configuring elements such as IAM and critical data storage as they set up cloud services as well as ensuring that they are deploying secure code.

This research paper aims to shed light on the security considerations in serverless environments and provide recommendations that can help ensure that serverless deployments are kept as secure as possible.

Serverless Architectures

Serverless computing refers to the technology that supports back-end services and allows enterprises to take advantage of shifting certain responsibilities to CSPs such as AWS, including capacity management, patching, and availability. With serverless computing, enterprises can build back-end applications without being directly involved in availability and scalability. Aside from the relative affordability of serverless computing, its architecture also allows enterprises to write and deploy code without worrying about the management and the security of the underlying infrastructure — hence the term “serverless.”

Because the infrastructural computing components of serverless technology — including the management of the hardware, the operating system, and the pieces of software — are handled by the CSPs themselves, these serverless components are also protected by their security. But although CSPs have a bigger slice of the responsibility pie for security with regard to serverless computing, the shared responsibility model³ also applies to serverless services. For example, in the case of a service categorized as infrastructure as a service (IaaS), the user has to implement all of the security measures to protect the operating system and the application software. When exposing any service to the internet, the user also has to manage their own firewall policy rules. The user is also responsible for its application and data.

In a serverless architecture, which is also referred to in terms of abstracted services, major CSPs like AWS ensure a secure infrastructure for its users so that they can focus on managing and securing their own data — including application code and binary data as well as asset classification and permission provision.

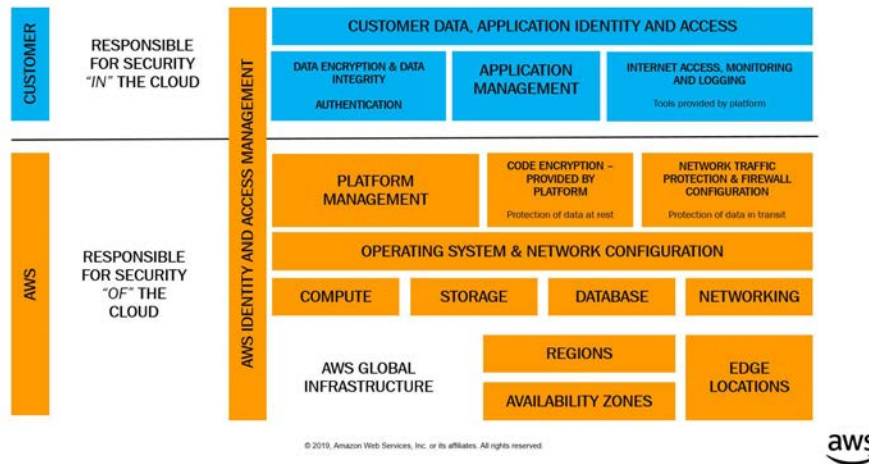
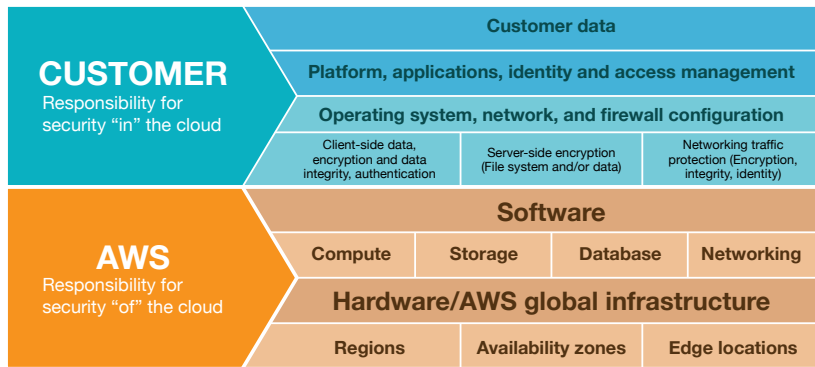


Figure 1. The classic shared responsibility model (top) and the shared responsibility model for AWS Lambda (bottom)

Image credits: AWS^{4, 5}

CSPs have two core design principles: least privilege⁶ and default deny. Least privilege provides a layer of security as most serverless applications are built on a heterogeneous set of services, while default deny allows for the purposeful interaction of the microservices that applications are built on. For their part in the shared responsibility model, administrators and users should follow these principles in designing and enforcing policies and permissions for the services used in their serverless applications.

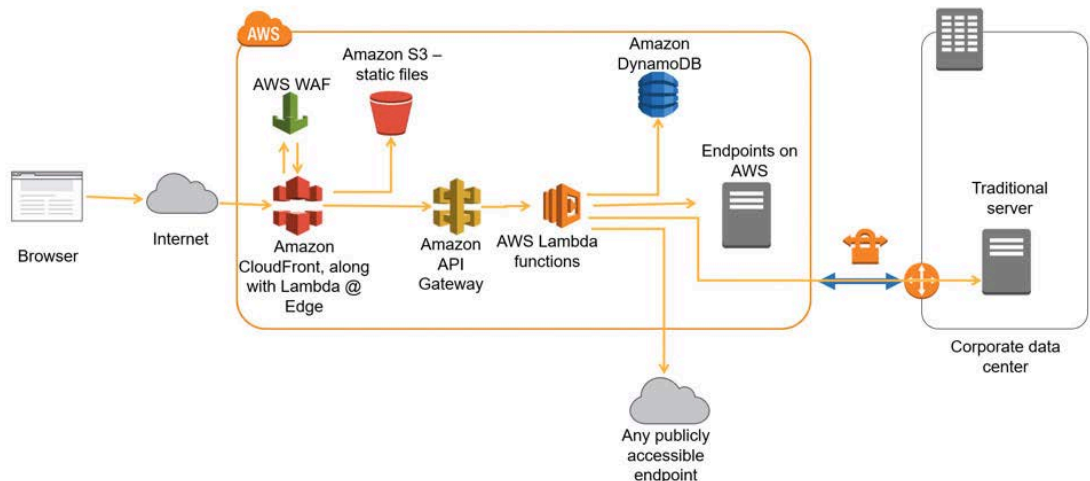


Figure 2. An example of a serverless web application pipeline

Image credit: AWS Compute Blog⁷

Connected Services in a Serverless Architecture

Understanding how a serverless architecture operates entails understanding the different services involved in it. The scope of this paper includes services offered by AWS, which is the top provider of serverless solutions.⁸ (According to a recent survey, more than 80% of developers are aware of AWS Lambda solutions while 51% use these solutions in their serverless deployments.⁹)

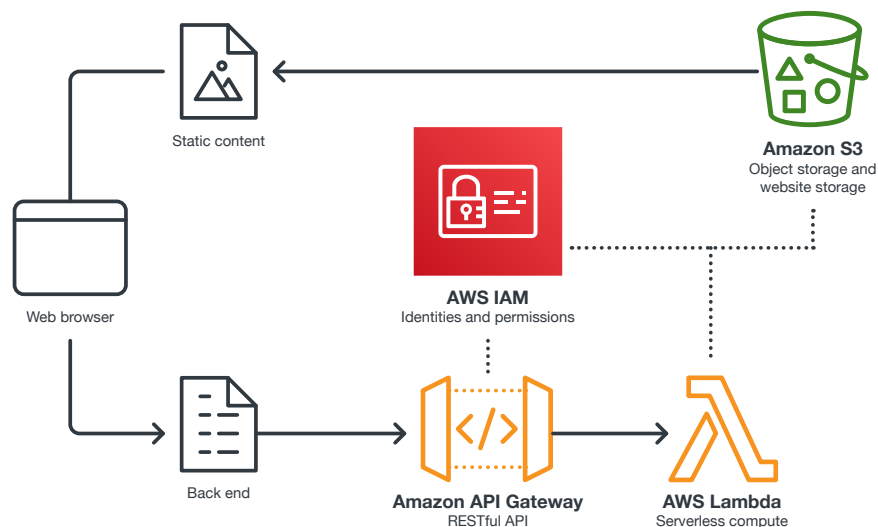


Figure 3. An example of interconnected services in a serverless architecture

Amazon S3

Amazon Simple Storage Service (Amazon S3)¹⁰ is an object storage service for a scalable amount of data that supports a variety of use cases, such as mobile applications, big data analytics, and internet-of-things (IoT) devices. Amazon S3 enables enterprises to manage objects, which are then stored in buckets via APIs.

When creating Amazon S3 buckets, a user is given the option of applying least privilege for their objects and buckets. An administrator with Amazon S3 permissions sets the policies for users, and users can be granted enough permissions to manage additional permissions or permissions can be locked down so that a bucket can be used only in certain ways. A user with permissions can manage a specific bucket in that region through the web console or the remote API. For both scenarios, the user has to be authenticated to be given access. However, if needed or desired, the user also has the option to make an object or a bucket available, depending on the policy set by the administrator. It can be made available to the public or to other internal services or users.

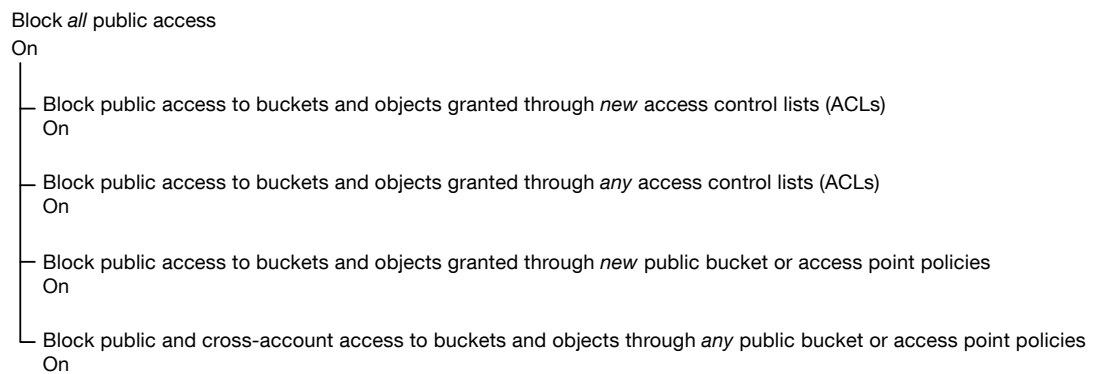


Figure 4. The default Amazon S3 bucket public access configuration

AWS Lambda

One of the most popular serverless services today,¹¹ AWS Lambda allows enterprises to run code without the hassle of server provisioning and maintenance. With it, developers can run code and pay only for the number of instances when the code is triggered. With a selected set of programming languages, AWS Lambda enables them to focus on their tasks without having to manage hardware or ensuring that the operating system and all of the installed applications are up to date. If an enterprise uses AWS Lambda solely, it can just focus on writing secure code. If AWS Lambda is used alongside another service, an enterprise has to be mindful of the permissions being granted. The user who defined the permissions must enforce the least-privilege approach for this service, which means that if AWS Lambda is going to work with other services, permissions must be granted manually.

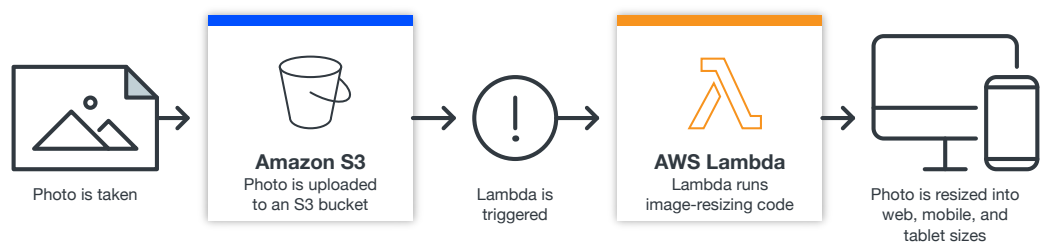


Figure 5. A use case for AWS Lambda

Amazon API Gateway

Amazon API Gateway is a service that enables easy and efficient creation, publishing, maintenance, monitoring, and securing of APIs.¹² As its name implies, it acts as a portal for applications, allowing them to access back-end service functionalities or data using RESTful APIs and WebSocket APIs. As with AWS Lambda's payment model, Amazon API Gateway allows enterprises to pay only for the API calls received and the amount of data transferred out. It can also act as a bridge that connects a deployment to other Amazon services.

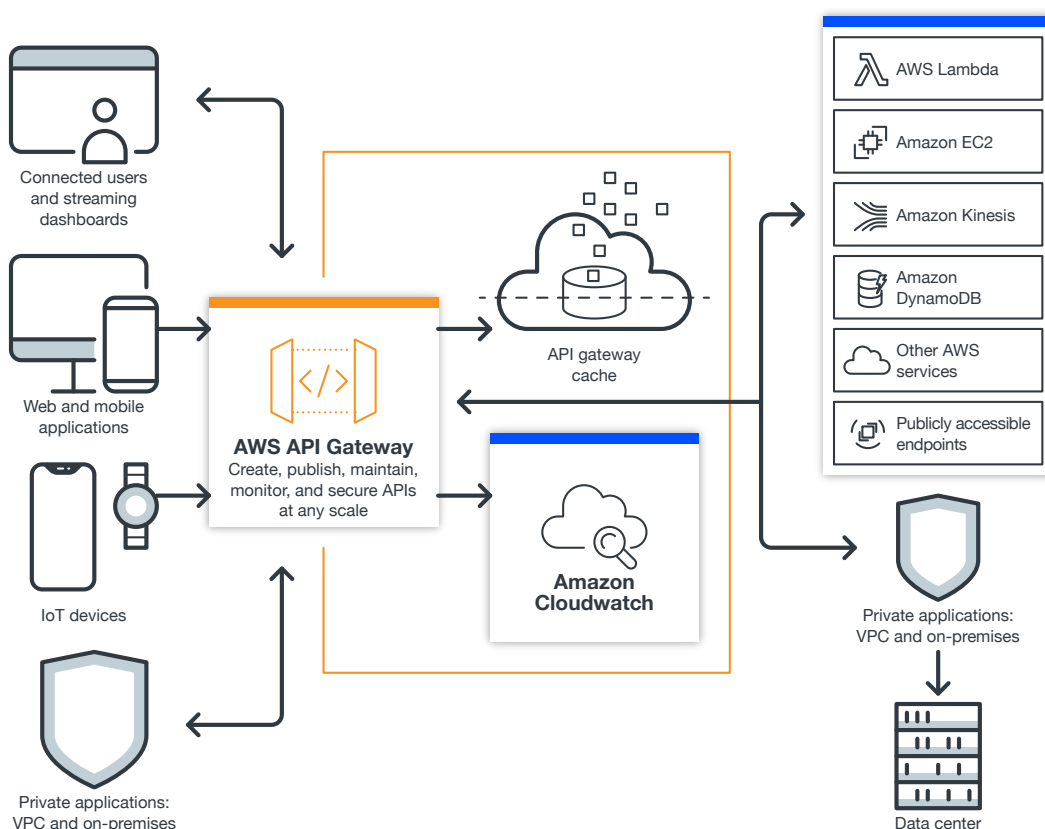


Figure 6. A use case for Amazon API Gateway

AWS IAM

AWS Identity and Access Management (AWS IAM) enables developers to create security credentials and permissions, and assign them to users. Using AWS IAM, an enterprise can also allow identity federation to enable existing identities within the enterprise to gain access to the AWS Management Console, call APIs, and access resources even without the creation of unique AWS IAM users. The user who creates the AWS IAM policies must define and enforce the least-privilege approach to make sure that services are not accessible unless access is explicitly granted by the user. IAM roles can also be used as a best practice. An IAM role is not uniquely linked to a single person and long-term credentials. Instead, it provides temporary credentials for a roles session.¹³

Misconfigurations and Unsecure Coding Practices

Users of the aforementioned services should define policies to use the least-privilege approach as a best practice, and should diligently assign and check privileges for a better security posture. However, a complex mix of services might prove difficult for users to manually address. In this section, we discuss and demonstrate misconfigurations and risks users might introduce or overlook when securing specific AWS serverless services. To help avoid these misconfigurations and risks, AWS provides its users with the AWS Well-Architected Framework,¹⁴ a set of architectural best practices for designing and running secure and efficient cloud systems.

Amazon S3

Amazon offers a comprehensive guide to keeping Amazon S3 buckets secure,¹⁵ including recommendations on enabling multi-factor authentication delete to avoid accidental deletions of buckets and individual objects within buckets. However, even with such guidance, users could still encounter security pitfalls such as the following.

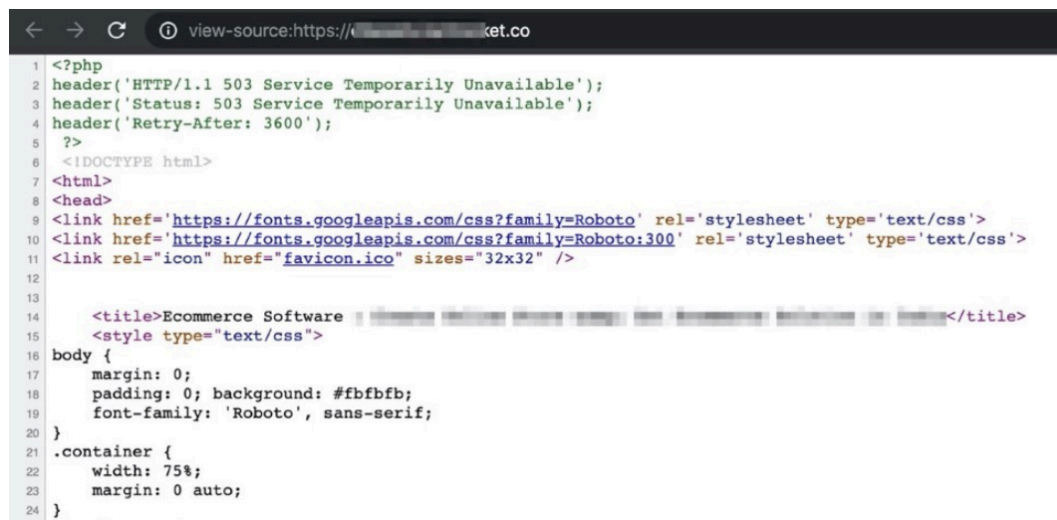
Leaving a Bucket Open and/or Publicly Accessible

Malicious actors could abuse an open or publicly accessible bucket to look for sensitive data. There have been cases in which critical and sensitive data has been left open because of misconfigurations, such as when a database containing more than 500,00 sensitive and private legal and financial documents was exposed.¹⁶ In February 2020, an exposed repository associated with a cloud-based application called JailCore was discovered leaking over 36,000 inmate records from various correctional facilities in the US.¹⁷ This has since been addressed by JailCore and the repository has been properly secured.

Unfortunately, many Amazon S3 buckets are left unsecured and open to the public despite recommended best practices. During our research, we were able to find open buckets that had sensitive data and buckets that were not completely open but were observed to be indexing accessible data.

AWS has always been clear on what Amazon S3 buckets are meant to host: static content. Amazon S3 should not be used to host dynamic content.¹⁸ Thus, server-side scripts should not be used. However, many users are not following the recommended use for Amazon S3 buckets and are opening themselves up to greater risks. It is not difficult to find websites hosted on Amazon S3 buckets that implement server-side script languages such as PHP, JSP, or ASP.NET. As it is not interpreted by the web server, the code is going to be exposed when the website's source code is accessed.

During our research, we found a webpage, hosted on an Amazon S3 bucket, that contains PHP code that is not being interpreted since it is not static. As a result, when the source code of this page is queried, anyone can see that the page uses command-line tools like *curl* or *wget*. This could expose sensitive data, such as credentials, or code parts that are not meant to be seen by the public, such as resource paths and programming logic. This could also make it easier for malicious actors to find and exploit vulnerabilities in the code by using techniques such as cross-site scripting (XSS) and SQL injection (SQLi).^{19, 20}



```
1 <?php
2 header('HTTP/1.1 503 Service Temporarily Unavailable');
3 header('Status: 503 Service Temporarily Unavailable');
4 header('Retry-After: 3600');
5 ?>
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <link href='https://fonts.googleapis.com/css?family=Roboto' rel='stylesheet' type='text/css'>
10 <link href='https://fonts.googleapis.com/css?family=Roboto:300' rel='stylesheet' type='text/css'>
11 <link rel="icon" href="favicon.ico" sizes="32x32" />
12
13
14 <title>Ecommerce Software </title>
15 <style type="text/css">
16 body {
17     margin: 0;
18     padding: 0; background: #fbfbfb;
19     font-family: 'Roboto', sans-serif;
20 }
21 .container {
22     width: 75%;
23     margin: 0 auto;
24 }
```

Figure 9. A webpage, hosted on an Amazon S3 bucket, that we found in the wild showing exposed PHP code

AWS Lambda

Aside from AWS security recommendations that tackle data protection, configuration, and compliance,²¹ users can also benefit from certain AWS Lambda functions that are aimed at protecting users' data as well as making sure costs do not get out of hand. However, users could still encounter security incidents because of unsecure practices such as the following.

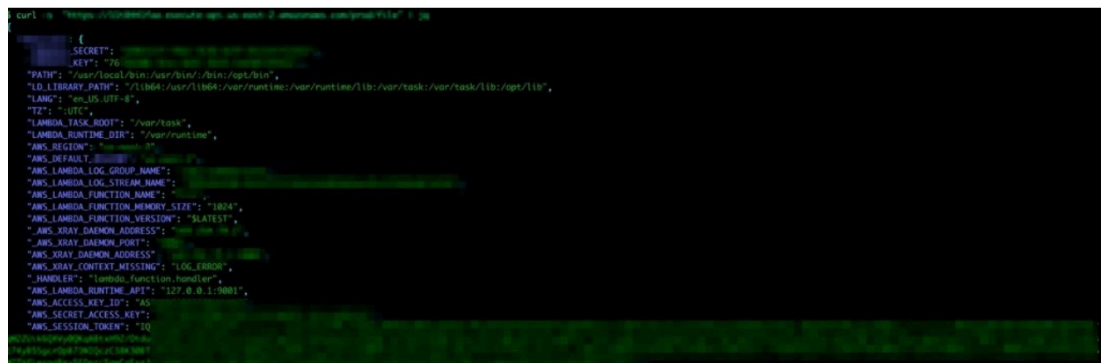
Creating Bad or Vulnerable Code

Since AWS Lambda functions are meant to be triggered to run code and deliver output, there are multiple scenarios where bad coding of an AWS Lambda function could allow a malicious actor to use common code injection techniques, such as when a function deals with user input data. In addition, the input data might not be coming from a regular browsing or HTTP API call. AWS Lambda functions could be deployed in a number of different scenarios, and malicious injections could be sent from different triggers, such as those from Amazon Simple Notification Service (SNS),²² email, and even the IoT. In such cases, a regular web application firewall (WAF) on the API level is not capable of protecting AWS Lambda functions against malicious injections.²³

Leaving Sensitive Data Exposed

When running AWS Lambda functions, there are a few environment variables that are created to pass environment-specific settings, such as authorizations, the size of resources, and the container host that is running the code. To authorize or manage other services or layers, users can create their own variables with users, passwords, authorization tokens, and application parameters. Since the container is going to expire after execution, data is not kept.

However, if the AWS Lambda function code is configured to return variables and is accessible from outside services, or if a malicious actor succeeds in injecting commands in the function, data could leak.



```
curl -s -XGET https://api.amazonaws.com/v1/lambda/arn:aws:lambda:us-east-1:123456789012:func:myfunc:latest?env
{
  "SECRETS": {
    "KEY": "75"
  },
  "PATH": "/usr/local/bin:/usr/bin:/bin:/opt/bin",
  "LD_LIBRARY_PATH": "/11864:/usr/11864:/var/runtime:/var/task:/lib:/usr/lib",
  "LANG": "en_US.UTF-8",
  "TZ": "UTC",
  "LAMBDA_TASK_ROOT": "/var/task",
  "LAMBDA_RUNTIME_DIR": "/var/runtime",
  "AWS_REGION": "us-east-1",
  "AWS_DEFAULT_REGION": "us-east-1",
  "AWS_LAMBDA_LOG_GROUP_NAME": "aws-logs-1-123456789012-us-east-1",
  "AWS_LAMBDA_LOG_STREAM_NAME": "2018-11-15T12:34:56.789Z",
  "AWS_LAMBDA_FUNCTION_NAME": "myfunc",
  "AWS_LAMBDA_FUNCTION_MEMORY_SIZE": "1024",
  "AWS_LAMBDA_FUNCTION_VERSION": "LATEST",
  "AWS_XRAY_DAEMON_ADDRESS": "127.0.0.1:9242",
  "AWS_XRAY_DAEMON_PORT": "9242",
  "AWS_XRAY_CONTEXT_MISSING": "LOG_ERROR",
  "HANDLER": "lambda_function.handler",
  "AWS_LAMBDA_RUNTIME_API": "127.0.0.1:9001",
  "AWS_ACCESS_KEY_ID": "AS",
  "AWS_SECRET_ACCESS_KEY": "AS",
  "AWS_SESSION_TOKEN": "TQ"
}
```

Figure 10. Variables shown from an AWS Lambda function output

Saving Credentials as Variables

In checking where AWS Lambda functions run inside a virtual machine, it is striking that the credentials and the secrets — the tokens and the keys used by functions for authentication — are being stored inside as variables. In a hypothetical malicious activity, if the actor manages to compromise a container and downloads, executes, and runs the AWS Command Line Interface (AWS CLI) tool,²⁴ they could gain access to a user's account without requesting any further credentials since the AWS CLI tool facilitates a one-time login.

Using Vulnerable Libraries

When developing a function with a preferred programming language, the user must keep in mind that the imported libraries as well as the code itself have to be secure. Using components with known vulnerabilities is one of the top 10 risks of web applications, according to the Open Web Application Security Project (OWASP).²⁵

Copying Bad Code Examples From Online Repositories

Developing an AWS Lambda function might be different from developing an application that would run inside an environment that the user has full control over. AWS provides essential documentation and basic starter codes to help users familiarize themselves with the service.²⁶ However, users who are looking for more complex or elaborate codes on online code sharing platforms should exercise caution prior to promoting such codes to production as these might have been developed using bad practices or could contain vulnerable components. Research has pointed to how shared code in online programming Q&A sites could be of poor quality and harbor vulnerabilities.²⁷

```
server_login = event["Records"][0]["Sns"]["MessageAttributes"]["ServerLogin"]["Value"]
print("Server Login: " + server_login)

server_password = event["Records"][0]["Sns"]["MessageAttributes"]["ServerPassword"]["Value"]
print("Server Password: " + server_password)

server_url = event["Records"][0]["Sns"]["MessageAttributes"]["ServerUrl"]["Value"]
print("Server URL: " + server_url)

# Format final protractor command
protractor_cmd = protractor_template.format(test_file_path, server_url, server_login, server_password,
                                           shell_safe_test_name)

time_remaining_after_test_timeout = 30 # Seconds
test_timeout = int(context.get_remaining_time_in_millis() / 1000) - time_remaining_after_test_timeout
print('Remaining seconds: ' + str(test_timeout))
print(protractor_cmd)

test_timed_out = False

console_output = '/tmp/console.log'

# TODO: Pass custom environment vars to prevent task from accessing anything sensitive
run_test = subprocess.Popen(protractor_cmd,
                             stderr=subprocess.STDOUT,
                             stdout=open(console_output, 'w'),
                             shell=True)

try:
    run_test.wait(test_timeout)
except:
    print('Test timed out')
    test_timed_out = True
    run_test.terminate()

test_stop_milli = current_time_milli()

print("###UIA TEST OUTPUT###")
print(run_test)
```

Figure 11. Sample code from GitHub

File Persistence

Files can be written in the `/tmp` folder inside an AWS Lambda execution environment. Files written there can have execution permission, which means that if a malicious actor successfully exploits a bad code, they could save their tools and scripts in this folder and execute them from there. Based on our tests, unless the code is changed on AWS Lambda's console, the files written there could last for as long as 12 minutes. A side "problem" is that if a container has access to the internet (through a fast internet connection, just for metrics' sake), a function configured with a 45-second timeout could download a 334-megabyte file. If an attacker is using the function as a pivoting point, this would allow the attacker to more easily download hacking tools for further use.

```
Interval time: 720
{
  "Date": [
    "Mon Apr 13 23:30:40 UTC 2020"
  ],
  "Uptime": [
    " 23:30:40 up 48 min,  0 users,  load average: 0.08, 0.02, 0.01"
  ],
  "Check if trashfile exists": 1,
  "Downloading a large file": [],
  "Create test file": [],
  "Hash file": [
    "df54b812d993784b9aa3906f797afbb60e28c29f /tmp/debian.iso"
  ],
  "List file data": [
    "--rw-rw-r-- 1 sbx_user1051 495 0 Apr 13 23:30 /tmp/Persistentfile"
  ]
}
{
  "Date": [
    "Mon Apr 13 23:43:02 UTC 2020"
  ],
  "Uptime": [
    " 23:43:02 up 55 min,  0 users,  load average: 0.00, 0.00, 0.00"
  ],
  "Check if trashfile exists": 1,
  "Downloading a large file": [],
  "Create test file": [],
  "Hash file": [
    "df54b812d993784b9aa3906f797afbb60e28c29f /tmp/debian.iso"
  ],
  "List file data": [
    "--rw-rw-r-- 1 sbx_user1051 495 0 Apr 13 23:43 /tmp/Persistentfile"
  ]
}
```

Figure 12. An example of a large file written in the `/tmp` folder inside an AWS Lambda function that is persistent for 12 minutes

Amazon API Gateway

AWS has created security features for Amazon API Gateway users to take advantage of when developing and implementing their security policies. AWS also provides helpful considerations for different types of enterprise environments.²⁸ Despite these, users could still unsafely expose Amazon API Gateway endpoints. One possible cause for this is users' reliance on default security.

Amazon API Gateway is a service that adds another layer that helps filter requests and prevent direct exposure of serverless services. Although it performs initial input filtering, its default configuration can benefit from improved security capabilities that could and should be implemented from the start, such as authentication. It should be noted, however, that while Amazon API Gateway provides additional protection, it should by no means be viewed as a replacement for a WAF.

An open and exposed Amazon API Gateway endpoint could be abused as the point of entry in an attack that seeks to compromise the service behind it, be triggered to cause a denial-of-service (DoS) attack,²⁹ or even raise an enterprise's bill if an AWS Lambda function is made to be queried frequently or incessantly.

AWS IAM

AWS provides guidelines on how to help better secure AWS resources that can be accessed via AWS IAM. However, users could still run into security holes because of the following unsecure practices.

Modifying a Service Using Credentials Saved as Variables and the AWS CLI Tool

As previously noted, credentials are stored inside the AWS Lambda infrastructure as variables. If a malicious actor succeeds in exploiting a function and uploads the AWS CLI tool, the AWS CLI tool could use the saved credentials to manage AWS services, including AWS Lambda itself.

Using Inadequate or Very Permissive Configurations

We often find projects on online code sharing platforms in which the developers use policies that are at the opposite end of the spectrum of the least-privilege policy — policies that are very permissive, thereby making sure that the whole system communicates and all services are responsive to one another. Since AWS IAM confirms the access to AWS services and resources, and thus dictates what services AWS Lambda can talk to, this example also falls under the considerations for AWS Lambda security — the copying of bad code examples from online repositories represents a security hole.

```
Policies:
- PolicyName:
  Fn::Sub: '${Namespace}-${AWS::Region}-iotClientRolePolicy'
  PolicyDocument:
    Version: "2012-10-17"
    Statement:
      # - Effect: Allow
      # Action:
      # - dynamodb:*
      # Resources:
      # - Fn::Sub: 'arn:aws:dynamodb:${AWS::Region}:${AWS::AccountId}:table/Bucket#Table'
      - Effect: Allow
      Action:
        - logs:CreateLogGroup
        - logs:CreateLogStream
        - logs:PutLogEvents
      Resource: arn:aws:logs::log-group:/aws/lambda/*:*:*
      # restrict client connect by client id
      - Effect: Allow
      Action: ["iot:Connect"]
      Resource:
        Fn::Join:
          - ""
          - - Fn::Sub: 'arn:aws:iot:${AWS::Region}:${AWS::AccountId}:client/'
            - "${iot:ClientId}"

policies:
- PolicyName:
  PolicyDocument:
    Version: '2012-10-17'
    Statement:
      - Action:
        - iam:*
        - logs:Delete*
      Effect: Allow
      Resource: '*'
      - Action:
        - ec2:*
        - rds:*
      Effect: Allow
      Resource: '*'
```

Figure 13. A code example found in the wild with a well-written policy implementing least privilege (left) and a code example found in the wild with a very permissive policy (right)

Depending on the size of a project, handling all of the policies and the permissions manually is almost humanly impossible as it can become too complicated and confusing. For this reason, enterprises with complex AWS workloads and applications should have a centralized auditing tool for IAM and for managing access keys, security credentials, and permission levels.³⁰ AWS IAM functionalities such as the access advisor³¹ and the Access Analyzer³² can also help in monitoring policies and permissions.

Possible Compromise and Attack Scenarios

AWS provides security mechanisms used in serverless services for users to set up and configure. However, malicious actors look for various ways to take advantage of common user errors, misconfigurations, and even one of the serverless model's own strengths — its distributed nature — so as to proceed with their activities.

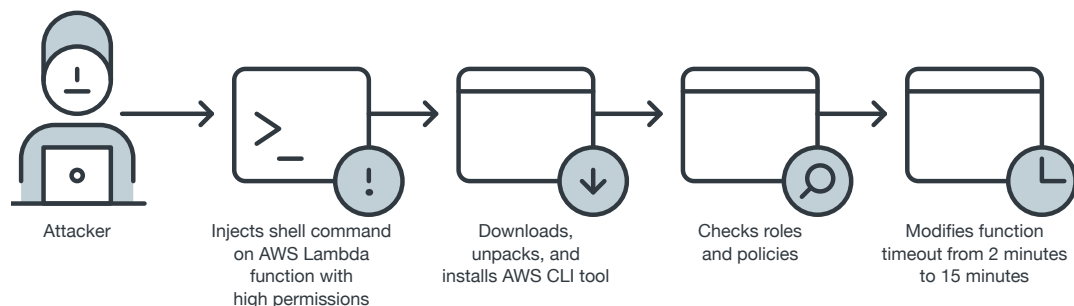


Figure 14. An attack chain involving an AWS Lambda function with high permissions

Credential and Account Theft

Functions, when accessing secure resources, need secrets. When a malicious actor gets ahold of a compromised application within a serverless system, they could take and use secrets containing secure credentials to gain access to critical resources or take control of the entire account — especially since secrets are converted into plain text when not in use.

Sensitive Data and Code Theft

When Amazon S3 buckets are made to host dynamic content for a web application, it could be easy for malicious actors to see sensitive data and critical parts of code by merely querying the site. This could allow them to use sensitive data, such as usernames and passwords, and command-line tools in performing reconnaissance or looking for vulnerabilities that they could exploit. When sensitive user data is exposed publicly, it could also cause reputational and financial harm to enterprises.

Privilege Escalation

When users do not properly define and configure permissions attached to a service, privilege escalation could happen. This could allow a compromised low-privileged user to change the password of a high-privileged user. The same goes for improperly configured role permission policies, which could allow a malicious user to create a new policy version that could in turn allow the changing of permissions in a policy. If role permissions are not securely set, the malicious user could be granted full administrator privileges. It is important to note that although it takes time to create a comprehensive list of roles that states which users and services in the architecture are allowed to pass, such a list helps ensure a more secure, misconfiguration-free system.

Misuse of Resources That Generates Cost for the Account Owner

Because serverless services such as AWS Lambda can provide resource elasticity and automated scalability, malicious actors who are looking to cause financial cost to enterprises for whatever reason could introduce code that would, for example, incessantly query AWS Lambda functions, thereby causing costs to go up. Malicious actors could basically change everything that is manageable through the AWS CLI tool, including the memory allocation of an AWS Lambda function, which could make costs balloon even more.

Persistence

Serverless functions have associated configuration information (name, description, entry point, and resource requirements) and are ephemeral, that is, they are given only a few seconds or minutes of life. They are also designed to be stateless, allowing for the quick launching of as many copies of a function as necessary to scale to the rate of incoming events. With each function call, a new instance of the function is created. The first time a function is called, known as a “cold start,” is expected to have a bit of latency. To avoid latency, functions need to remain cached — also known as a “warm start” — wherein the same function sandbox or container for different invocations are reused. Warm starts might be taken advantage of by malicious actors who have already compromised an AWS Lambda instance to periodically send requests so as to prevent the instance from being taken down.

Other Security Considerations in Serverless Deployments

Amazon provides security features for their services and guidance for users to consider in order to secure their respective environments. But it is safe to say that there will always be room for better security for serverless services. In this section, we discuss opportunities for improving security in the connected services of a serverless architecture. (We recommend best practices and other security measures that address these issues in the next section.)

Amazon S3

To keep Amazon S3 buckets secure, new Amazon S3 buckets are generated with public access blocked by default.³³ AWS documentation also points to strong security recommendations that users can apply to ensure that their Amazon S3 buckets are protected using AWS configurations.³⁴ These include making sure buckets are private, implementing authentication protocols, and adding more layers of protection to buckets to further restrict who can access them from every point of entry.

With regard to troubleshooting Amazon S3 bucket abuse, something that makes the process more challenging is the lack of a feature for easy logging. The available default logging option supports basic storage consumption and usage, but it does not allow users to see who is accessing their Amazon S3 buckets. What users can do is to turn on object-level logging, which logs Amazon S3 object-level API operations such as *GetObject*, *DeleteObject*, and *PutObject* to AWS CloudTrail.³⁵

In addition, if a user is hosting a website on an Amazon S3 bucket, there is an option to log user access and send all logs to a dedicated folder inside an Amazon S3 bucket. This is a good option that can address the troubleshooting difficulty, but it can also create a large number of files that contain fragmented process logs. For better visibility, users can use a serverless interactive query service for analyzing data (like Amazon Athena³⁶) and a cloud security posture management service with extensive reporting capabilities.


```
$ curl -sv [redacted]
* Trying [redacted]
* TCP_NODELAY set
* Connected to [redacted] port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/cert.pem
*   Cpath: none
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
* ALPN, server accepted to use h2
* Server certificate:
*  subject: CN=*.execute-api.us-east-2.amazonaws.com
*  start date: Sep 27 00:00:00 2019 GMT
*  expire date: Oct 27 12:00:00 2020 GMT
*  subjectAltName: host "[redacted]" matched cert's "*.execute-api.us-east-2.amazonaws.com"
*  issuer: C=US; O=Amazon; OU=Server CA 1B; CN=Amazon
*  SSL certificate verify ok.
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade: len=0
* Using Stream ID: 1 (easy handle 0x7fb1eb00c400)
* GET /[redacted] HTTP/2
* Host: [redacted].amazonaws.com
* User-Agent: curl/7.64.1
* Accept: */*
*
* Connection state changed (MAX_CONCURRENT_STREAMS == 128)!
< HTTP/2 200
< date: Mon, 06 Apr 2020 19:19:41 GMT
< content-type: application/json
< content-length: 4861
< x-amzn-requestid: [redacted]
< x-amz-apigw-id: K1[redacted]
< x-amzn-trace-id: R[redacted]
```

Figure 16. An HTTP header showing an Amazon S3 bucket's *x-amzn-RequestId* and *x-amz-apigw-id* information

AWS IAM

As a user's environment grows, the policies and the roles also increase in complexity. Tracking and managing these could prove challenging without the aid of an external auditing tool.

Security Measures for Serverless Services

Serverless services have become essential business tools, and keeping them secure should be of utmost priority not just for enterprises that use them but for the people and the organizations relying on these enterprises' applications as well. The following are some best practices and security solutions that can help keep serverless services secure.

Amazon S3

It is safe to say that AWS does its part on the shared responsibility model. It also helps users fulfill their part of the model by making it more difficult for administrators to expose their buckets without knowing what they are doing. Thus, administrators need to constantly check whether configurations are properly set by using services such as AWS Config and AWS CloudTrail, or by querying the Amazon S3 APIs. They can also use third-party tools that scan the contents of Amazon S3 buckets and their policies to check access and configurations for both security and compliance³⁸ — such as DNS (Domain Name System) compliance of Amazon S3 bucket names, public read/write access, and encryption of data via TLS (Transport Layer Security) during transport.

AWS Lambda

AWS makes sure that the execution environments used in AWS Lambda functions are ephemeral. This means that execution environments are constantly being fully replaced by brand-new ones. Because of this, saved data inside a container will remain there only for a short period, making it more difficult for malicious actors to perform privilege escalation or lateral movement.

Another noteworthy AWS Lambda feature is the function timeout.³⁹ It is aimed at protecting users from malicious activities that cause unintentionally high billing. Each programming language has its own default function timeouts, which are usually set quite low. For example, when creating an app using Python, a user is given 6 seconds before an instance expires. This is cost-effective and serves as a security feature: The less time an instance has, the less

time an attacker has to carry out an attack. However, building an application with a number of features, using nonoptimized code, and adding layers such as security features to the code require increasing the function timeouts, which could give a malicious actor more time to work and compromise an application.

While it is far from being an all-in-one cybersecurity solution, as in the case of malicious injection attacks, a WAF can still provide additional filtering for data that is being passed through AWS Lambda functions as an extra security measure. In general, a WAF can provide good security based on a list of disallowed web request conditions, such as the presence of SQL code or a script that is likely to be malicious, and a list that allows the entry of data from trusted sources.

Users can also benefit from solutions that closely monitor AWS Lambda functions to ensure that functions do not run with administrator privileges and are not exposed to the public, and that tracing is enabled for functions.⁴⁰

Amazon API Gateway

Although IT environments vary in size and needs, Amazon provides the following security recommendations for Amazon API Gateway users:⁴¹

- Implement the least-privilege policy when creating, reading, updating, or deleting APIs.
- Monitor REST APIs and log API requests using Amazon CloudWatch⁴² and Amazon Kinesis Data Firehose.⁴³
- Enable AWS CloudTrail, an AWS service that keeps a detailed record of the actions taken by a user, a role, or another AWS service in Amazon API Gateway for better monitoring.⁴⁴
- Enable AWS Config, which enables users to see the configuration of all AWS resources in their systems and check how the resources are connected, and to see the configuration history.⁴⁵

On top of these, users can turn on AWS WAF,⁴⁶ which integrates with Amazon API Gateway, or use third-party solutions to protect Amazon API Gateway APIs from common web exploits. Audit tools can help ensure that best practices for Amazon API Gateway are followed (for example, that APIs have content encoding enabled).⁴⁷

AWS IAM

Amazon provides a detailed list of security best practices for AWS IAM that discusses various topics that can help protect AWS resources.⁴⁸ These are only a few:

- **Locking away the AWS account root user access key.** Because permissions cannot be reduced for the user access key, whoever has access to it has access to all AWS services.
- **Creating individual AWS IAM users.** An administrator can create individual users for accessing the AWS account so as to avoid giving away the AWS account root user credentials to anyone else.
- **Using groups to assign permissions to AWS IAM users.** User groups can be created according to job functions, with each group having its own level of access or set of relevant permissions.
- **Using roles to delegate permissions.** Instead of sharing security credentials between accounts, administrators should create IAM roles with already defined permissions and designate the roles only to AWS accounts with users who need to assume those roles.
- **Granting least privilege.** Administrators should grant only the permissions required for a task to be completed. They should map out what tasks certain users and roles need to do and design policies that allow these users and roles to do only those specific tasks.

Stronger security for AWS IAM also entails a solution that can ensure that AWS IAM access keys are rotated periodically, detect any and all configuration changes, and delete any unused users and groups.⁴⁹

Recommendations for Creating Secure Serverless Applications

CSPs such as AWS handle the security and the upkeep of the infrastructure of serverless services, and provide users with ephemeral and stateless functions. But the applications that users build and run still need to be made as secure as possible — especially since serverless applications can be made up of a large number of functions. The following are a few security recommendations that developers can consider when building serverless applications:

- **Establish a good code review process.** The creation of secure code falls under the responsibility of users, not CSPs. A solid code review process is critical in ensuring that applications are kept secure from the design stage to deployment.

- **Create better permissions compliance procedures.** Managing all of the policies and the permissions in a large-scale project can be overwhelming, especially when it is being done manually. For users to better manage complex permissions compliance procedures, they need to have a tool that runs checks against security best practices and industry compliance standards in an automated manner.
- **Avoid leaving footprints.** Most attacks, targeted or otherwise, start with a reconnaissance step. This is why it is a good security measure to minimize the number of footprints left by the services being used. For example, instead of deploying applications that directly expose Amazon API Gateway endpoints, users of Amazon API Gateway can opt for a native or third-party load balancer, a content delivery network (CDN), or a proxy.
- **Use application security solutions that protect web applications against application layer attacks.** Under the serverless computing model, traditional security that normally applies to data centers or workloads cannot be deployed in the operating system. To further strengthen security, users can take advantage of application security solutions that can automatically detect exploit attempts to prevent hacks and identify vulnerabilities, and protect the system against code vulnerabilities and data exfiltration on the server level, and other vulnerability attacks at the application level.⁵⁰

Serverless Technology and Shared Responsibility

In the serverless model, CSPs are in charge of securing critical software and hardware infrastructures. In the shared responsibility model, this is known as “security *of* the cloud.” CSPs also handle the security of other critical components, such as server-side encryption, the operating system, and network and firewall configuration. These are components that are part of “security *in* the cloud,” which in other cloud models falls under the users’ responsibility. CSPs also implement the least-privilege policy and the default-deny approach to service communications. In the case of AWS Lambda, AWS takes care of the underlying infrastructure, the operating system, and the application platform, while the users themselves are responsible for the security of their code, the storage and the accessibility of sensitive data, and the IAM in relation to AWS Lambda and within their function.

But all this should not make adopters of serverless services complacent about security. Maintaining proper configurations and code quality is imperative in keeping serverless services as secure as possible. We have shown in this research paper how malicious actors could take advantage of misconfigurations and errors to launch attacks on serverless deployments. Because of common user mistakes, malicious actors would not even need to do much to carry out attacks. In some cases, they could simply query an Amazon S3 bucket that hosts dynamic (rather than static) content to see sensitive data and critical parts of code. Misconfigurations and permissive policies could also allow privilege escalation to occur, enabling malicious actors to change permissions in policies or even take hold of administrator privileges.

Aside from security loopholes that arise from neglect or oversight, malicious actors could also take advantage of the features of serverless services. For instance, warm starts in function calls, which are meant to reduce latency, could be abused by malicious actors to launch stealthy and persistent attacks. Malicious actors could also abuse the elastic and automated scalability offered by serverless services by introducing code that could continuously query AWS Lambda functions, thereby depleting enterprises’ financial resources.

The security of serverless services can be further enhanced through the implementation of certain measures. Connected serverless services can benefit from security features such as an improved default logging option for Amazon S3 and a default authentication configuration for Amazon API Gateway endpoints. Users would do well to follow our security recommendations to defend serverless deployments from threats and risks. Developers of security solutions, for their part, should consider our guidance on vital features that security solutions should have in order to properly protect serverless deployments from vulnerabilities, hacks, misconfigurations, and other threats.

While it is true that the infrastructural computing components of serverless services are covered by CSPs, this does not mean that the users of the services are immune to making misconfigurations, whether due to negligence or lack of awareness. Users need to understand that the shared responsibility model also applies to serverless technology. Adopters of serverless services are responsible for keeping the critical components of serverless services as secure as possible by following secure coding practices, keeping secrets safe, monitoring and logging functions, configuring policies properly, and implementing the principles of least privilege and default deny. By detailing the possible compromise and attack scenarios among the connected services in a serverless architecture, we aim to highlight how important diligence in applying security practices is in thwarting threats and mitigating risks.

References

- 1 Trend Micro. (Oct.24, 2019). *Trend Micro Security News*. “The Cloud: What it is and what it’s for.” Accessed on May 25, 2020, at <https://www.trendmicro.com/vinfo/us/security/news/security-technology/the-cloud-what-it-is-and-what-it-s-for>.
- 2 Amazon Web Services. (March 22, 2019). *YouTube*. “Build a Serverless Startup in Just 30 Minutes!” Accessed on May 25, 2020, at <https://www.youtube.com/watch?v=qBNYmYRITpU>.
- 3 Mark Nunnikhoven. (Oct. 22, 2019). *Trend Micro Simply Security*. “The Shared Responsibility Model.” Accessed on May 25, 2020, at <https://blog.trendmicro.com/the-shared-responsibility-model/>.
- 4 AWS. (n.d.). AWS. “Shared Responsibility Model.” Accessed on May 25, 2020, at <https://aws.amazon.com/compliance/shared-responsibility-model/>.
- 5 AWS. (n.d.). AWS. “The Shared Responsibility Model.” Accessed on May 25, 2020, at <https://docs.aws.amazon.com/whitepapers/latest/security-overview-aws-lambda/the-shared-responsibility-model.html>.
- 6 Security Best Practices in IAM. (n.d.). AWS. “Security Best Practices in IAM.” Accessed on May 25, 2020, at <https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege>.
- 7 Chris Munns. (July 23, 2018). *AWS Compute Blog*. “Powering HIPAA-compliant workloads using AWS Serverless technologies.” Accessed on June 8, 2020, at <https://aws.amazon.com/blogs/compute/powering-hipaa-compliant-workloads-using-aws-serverless-technologies/>.
- 8 Ihor Lobastov. (March 8, 2019). *DZone*. “Comparing Serverless Architecture Providers: AWS, Azure, Google, IBM, and Other FaaS Vendors.” Accessed on June 3, 2020, at <https://dzone.com/articles/comparing-serverless-architecture-providers-aws-az>.
- 9 David Ramel. (May 8, 2020). *Virtualization and Cloud Review*. “Cloud-Native Development Survey Details Kubernetes, Serverless Data.” Accessed on May 25, 2020, at <https://virtualizationreview.com/articles/2020/05/08/cloud-native-dev-survey.aspx>.
- 10 AWS. (n.d.). AWS. “Amazon S3.” Accessed on May 25, 2020, at <https://aws.amazon.com/s3/>.
- 11 AWS. (n.d.). AWS. “Amazon Lambda.” Accessed on May 25, 2020, at <https://aws.amazon.com/lambda/>.
- 12 Amazon API Gateway. (n.d.). AWS. “Amazon API Gateway.” Accessed on May 25, 2020, at <https://aws.amazon.com/api-gateway/>.
- 13 AWS (n.d.). AWS. “IAM Roles.” Accessed on Aug. 11, 2020, at https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html
- 14 Derek Belt. (May 15, 2018). *AWS Partner Network (APN) Blog*. “The 5 Pillars of the AWS Well-Architected Framework.” Accessed on July 23, 2020, at <https://aws.amazon.com/blogs/apn/the-5-pillars-of-the-aws-well-architected-framework/#:~:text=The%20AWS%20Well%20Architected%20Framework%20provides%20architectural%20best%20practices%20across,an%20existing%20or%20proposed%20architecture>.
- 15 AWS. (n.d.). AWS. “Security Best Practices for Amazon S3.” Accessed on May 25, 2020, at <https://docs.aws.amazon.com/AmazonS3/latest/dev/security-best-practices.html>.
- 16 VPNMentor. (March 19, 2020). *VPNMentor*. “Report: Two Corporate Finance Companies Leak Half a Million Legal and Financial Documents Online.” Accessed on May 25, 2020, at <https://www.vpnmentor.com/blog/report-mca-wizard-leak/>.
- 17 Trend Micro. (Feb. 12, 2020). *Trend Micro Security News*. “Misconfigured AWS S3 Bucket Leaks 36,000 Inmate Records.” Accessed on May 25, 2020 at <https://www.trendmicro.com/vinfo/ph/security/news/cybercrime-and-digital-threats/misconfigured-aws-s3-bucket-leaks-36-000-inmate-records>.
- 18 AWS. (n.d.). AWS. “Hosting a static website on Amazon S3.” Accessed on May 25, 2020, at <https://docs.aws.amazon.com/AmazonS3/latest/dev/WebsiteHosting.html>.
- 19 Trend Micro. (n.d.). *Trend Micro*. “Cross-site scripting (XSS).” Accessed on May 25, 2020, at [https://www.trendmicro.com/vinfo/us/security/definition/cross-site-scripting-\(xss\)](https://www.trendmicro.com/vinfo/us/security/definition/cross-site-scripting-(xss)).
- 20 Trend Micro. (n.d.). *Trend Micro*. “SQL injection.” Accessed on May 25, 2020, at <https://www.trendmicro.com/vinfo/us/security/definition/sql-injection>.
- 21 AWS. (n.d.). AWS. “Security in AWS Lambda.” Accessed on May 25, 2020, at <https://docs.aws.amazon.com/lambda/latest/dg/lambda-security.html>.
- 22 AWS. (n.d.). AWS. “Amazon Simple Notification Service.” Accessed on May 25, 2020, at <https://aws.amazon.com/sns/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc>.
- 23 OWASP. (n.d.). OWASP. “Web Application Firewall.” Accessed on May 25, 2020, at https://owasp.org/www-community/Web_Application_Firewall.
- 24 AWS. (n.d.). AWS. “AWS Command Line Interface.” Accessed on May 25, 2020, at <https://aws.amazon.com/cli/>.

- 25 OWASP. (n.d.). OWASP. "OWASP Top Ten." Accessed on June 3, 2020, at <https://owasp.org/www-project-top-ten/>.
- 26 AWS. (n.d.). AWS. "AWS Lambda Documentation." Accessed on May 25, 2020, at https://docs.aws.amazon.com/lambda/?id=docs_gateway.
- 27 Morteza Verdi et al. (n.d.). *Arxiv.org*. "An Empirical Study of C++ Vulnerabilities in Crowd-Sourced Code Examples." Accessed on May 25, 2020, at <https://arxiv.org/pdf/1910.01321.pdf>.
- 28 AWS. (n.d.). AWS. "Security Best Practices in Amazon API Gateway." Accessed on June 4, 2020, at <https://docs.aws.amazon.com/apigateway/latest/developerguide/security-best-practices.html>.
- 29 Trend Micro. (n.d.). *Trend Micro*. "Denial of Service (DOS)." Accessed on May 25, 2020, at <https://www.trendmicro.com/vinfo/us/security/definition/denial-of-service-dos>.
- 30 Trend Micro Cloud Conformity. (n.d.). *Trend Micro Cloud Conformity*. "AWS IAM Best Practices." Accessed on May 25, 2020, at <https://www.cloudconformity.com/knowledge-base/aws/IAM/>.
- 31 AWS. (Dec. 7, 2018). AWS. "Automate AWS IAM Permissions Analysis Using the New IAM Access Advisor APIs." Accessed on Aug. 11, 2020, at https://aws.amazon.com/about-aws/whats-new/2018/12/iam_access_advisor_apis/.
- 32 AWS. (n.d.). AWS. "What Is AWS IAM Access Analyzer?" Accessed on Aug. 11, 2020, at <https://docs.aws.amazon.com/IAM/latest/UserGuide/what-is-access-analyzer.html>.
- 33 AWS. (n.d.). AWS. "Using Amazon S3 block public access." Accessed on July 23, 2020, at <https://docs.aws.amazon.com/AmazonS3/latest/dev/access-control-block-public-access.html>.
- 34 AWS. (n.d.). AWS. "Security Best Practices for Amazon S3." Accessed on May 25, 2020, at <https://docs.aws.amazon.com/AmazonS3/latest/dev/security-best-practices.html>.
- 35 AWS. (n.d.). AWS. "How do I enable object-level logging for an S3 bucket with AWS CloudTrail data events?" Accessed on July 30, 2020, at <https://docs.aws.amazon.com/AmazonS3/latest/user-guide/enable-cloudtrail-events.html>.
- 36 AWS. (n.d.). AWS. "Amazon Athena." Accessed on June 8, 2020, at <https://aws.amazon.com/athena/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc>.
- 37 Morton Swimmer et al. (April 8, 2020). *Trend Micro Security News*. "Exploring Common Threats to Cloud Security." Accessed on May 25, 2020, at <https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/exploring-common-threats-to-cloud-security>.
- 38 Trend Micro Cloud Conformity. (n.d.). *Trend Micro Cloud Conformity*. "AWS S3 Best Practices." Accessed on May 25, 2020, at <https://www.cloudconformity.com/knowledge-base/aws/S3/>.
- 39 AWS. (n.d.). AWS. "AWS Lambda Limits." Accessed on May 25, 2020, at <https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html>.
- 40 Trend Micro Cloud Conformity. (n.d.). *Trend Micro Cloud Conformity*. "Cloud Conformity Lambda." Accessed on May 25, 2020, at <https://www.cloudconformity.com/knowledge-base/aws/Lambda/>.
- 41 AWS. (n.d.). AWS. "Security best practices in Amazon API Gateway." Accessed on May 25, 2020, at <https://docs.aws.amazon.com/apigateway/latest/developerguide/security-best-practices.html>.
- 42 AWS. (n.d.). AWS. "Amazon Cloudwatch." Accessed on May 25, 2020, at <https://aws.amazon.com/cloudwatch/>.
- 43 AWS. (n.d.). AWS. "Amazon Kinesis Data Firehose." Accessed on May 25, 2020, at <https://aws.amazon.com/kinesis/data-firehose/>.
- 44 AWS. (n.d.). AWS. "AWS CloudTrail." Accessed on May 25, 2020, at <https://aws.amazon.com/cloudtrail/>.
- 45 AWS. (n.d.). AWS. "AWS Config." Accessed on May 25, 2020, at <https://aws.amazon.com/config/>.
- 46 AWS. (n.d.). AWS. "AWS WAF - Web Application Firewall." Accessed on Aug. 11, 2020, at <https://aws.amazon.com/waf/>.
- 47 Trend Micro Cloud Conformity. (n.d.). *Trend Micro Cloud Conformity*. "Amazon API Gateway Best Practices." Accessed on May 25, 2020, at <https://www.cloudconformity.com/knowledge-base/aws/APIGateway/>.
- 48 AWS. (n.d.). AWS. "Security Best Practices in IAM." Accessed on May 25, 2020, at <https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html>.
- 49 Trend Micro Cloud Conformity. (n.d.). *Trend Micro Cloud Conformity*. "AWS IAM Best Practices." Accessed on May 25, 2020, at <https://www.cloudconformity.com/knowledge-base/aws/IAM/>.
- 50 Trend Micro. (n.d.). *Trend Micro*. "Trend Micro Cloud One Application Security." Accessed on May 25, 2020, at https://www.trendmicro.com/en_ph/business/products/hybrid-cloud/cloud-one-application-security.html.



```
True  
False  
True  
False  
"ROR_Z":  
False  
False  
True
```

```
nd -add back the deselected mirror modifier object
```

```
ts.active = modifie  
modifier_ob)) # mod  
0  
ted_objects[0]  
[...]  
[...]
```

TREND MICRO™ RESEARCH

Trend Micro, a global leader in cybersecurity, helps to make the world safe for exchanging digital information.

Trend Micro Research is powered by experts who are passionate about discovering new threats, sharing key insights, and supporting efforts to stop cybercriminals. Our global team helps identify millions of threats daily, leads the industry in vulnerability disclosures, and publishes innovative research on new threat techniques. We continually work to anticipate new threats and deliver thought-provoking research.

www.trendmicro.com



**TREND
MICRO™**

research 

© 2020 Trend Micro Incorporated and/or its affiliates. All rights reserved. Trend Micro and the t-ball logo are trademarks or registered trademarks of Trend Micro and/or its affiliates in the US and other countries. Third-party trademarks mentioned are the property of their respective owners.