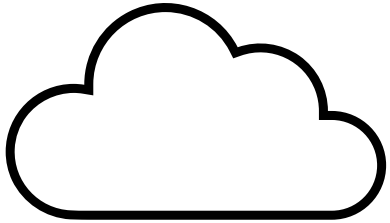
Security testing of cloud environments

Grega Prešeren

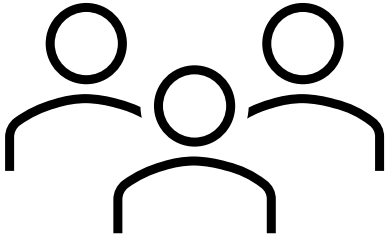
Key security issues in cloud enviroments

Responsibility for security is shared

- Cloud
  - Secures the infrastructure
  - Provides security features or tools



- You
  - Secure what you host on that infrastructure
  - Pay for and properly utilize security features or tools

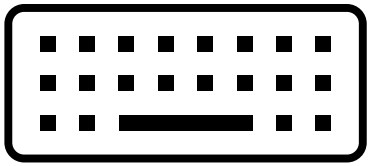


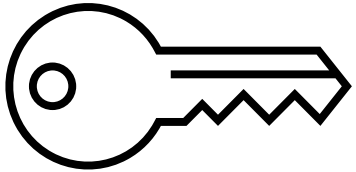
IAM

Root cause for majority of cloud breaches

IAM scope

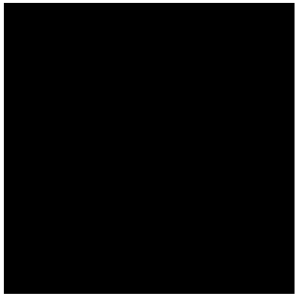
- Users/humans
  - Credentials
  - Phishing
- Software components
  - Access keys, roles, policies
  - Software exploits

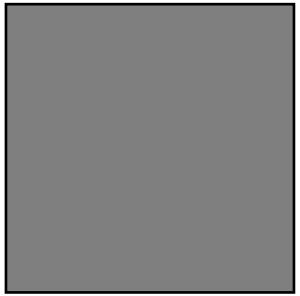


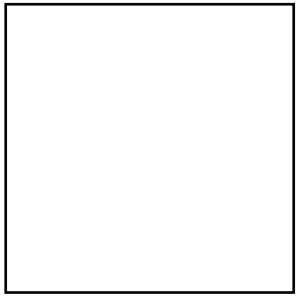


Security testing approaches

The box model of (security) testing

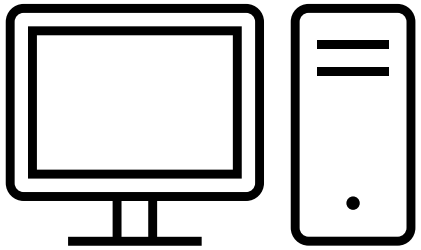




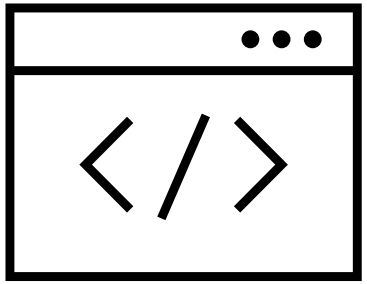


Human engagement

- Automated
  - Scanning
  - Exploiting
  - Part of CI/CD

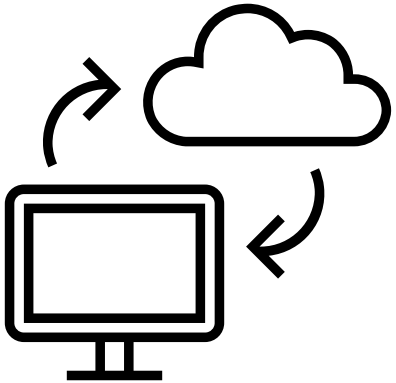


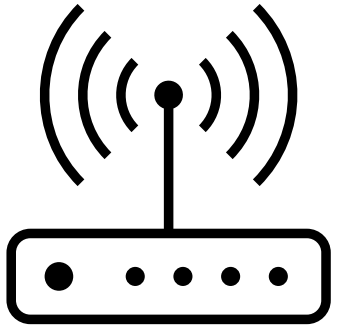
- Manual
  - False positives elimination
  - Custom exploiting



Level of access

- External
  - Attack surface identification?
- Internal
  - What is cloud internal?





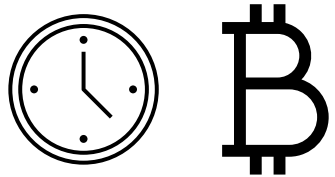
Adversarial realism

- Vulnerability assessment
  - Problem of false positives
- Penetration testing
  - Exploiting vulnerabilities & proving potential impact
- Red teaming
  - Full adversarial emulation
- Purple teaming
  - Red & blue team

How to test security of cloud environments?

Downsides of each testing type

- Black-box
  - A lot can be missed (i.e. false negative)
- Grey-box
  - Some false positives
    - Services with uncompliant settings not used at all
    - Additional features might provide the same or equivalent functionality
- White-box
  - Takes a lot of time
    - Tester's time to check everything
    - Customer's time to support the tester with explanations
  - Very expensive



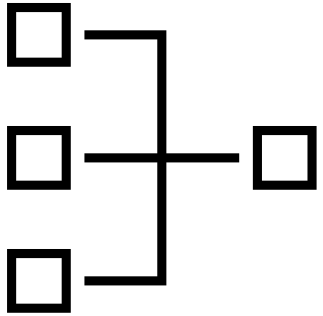
Benefits of not testing black-box

- Think for the future
  - What if one admin makes a mistake in key configuration, is there a complentary control in place?
- Defense in depth
  - If one security control fails, is there another one doing complementary job
- Compliance
  - An auditor need to be competent to get relevant results
  - It is not just paper work, if done correctly

How to secure cloud-native apps?

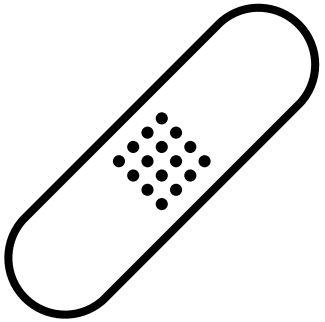
Cloud-native web app architecture

- Architect like it was not cloud-native (use old/legacy knowledge)
  - Service exposure to the public
  - Use encrypted protocols (HTTPS)
  - Secure public logins (password policy, MFA, etc.)
  - Authorization!



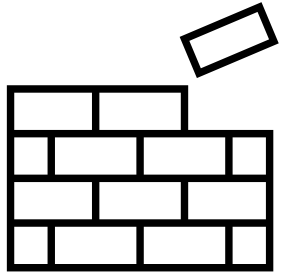
Patching in the cloud

- Patch everything ASAP
  - If applicable to your cloud services
    - EC2 kernel, OS-level patches, server software patches
    - Self-managed databases
    - RDS databases (major upgrades require approval)
  - Container images
  - Third-party libraries patching
    - jQuery, etc.



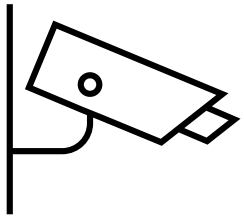
Leverage cloud security services

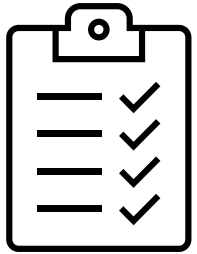
- Some basic security features are free
  - VPC – private networking (not accessible from Internet unless NAT is used)
  - Network ACLs – stateless firewall
  - Security Groups - stateful firewall
  - IAM users, roles, policies, and MFA
- Many security features need to be paid
  - AWS WAF – web app protection
  - AWS Firewall Manager – simplified security services management
  - AWS Secrets Manager – auto-rotate credentials (DB passwords, API keys, etc.)



Compliance services

- Monitoring and alerting
  - CloudTrail ($$) – log collector
  - GuardDuty ($$) – threat detection based on logs
  - CloudWatch ($$) – alerts based on logs
- Configuration hardening
  - AWS Config ($$) – manage configuration changes
  - Amazon Inspector ($$) – vulnerability scanning
  - AWS Security Hub ($$) – security posture dashboard
    - Can check compliancy with CIS AWS Foundations Benchmark
  - Independent benchmark check using external tools

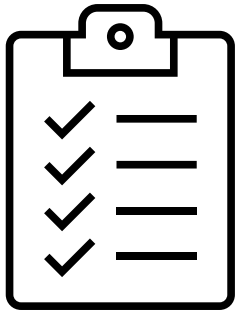




AWS real world examples

CIS Amazon Web Services Foundations Benchmark

- v7.0.0 - 03-25-2026
- Audit areas
  - Identity and access management
  - Storage
  - Logging
  - Monitoring
  - Networking



2.10 Ensure multi-factor authentication (MFA) is enabled for all IAM users that have a console password

- Mitigates susceptibility to credential compromise

- Identify users without MFA enabled
  - aws iam generate-credential-report
  - aws iam get-credential-report --query 'Content' --output text | base64 -d | cut -d, -f1,4,8

2.12 Ensure access keys are rotated every 90 days or less

- Access key is comprised of
  - Access key ID
  - Secret access key
- By default remain valid until manually deactivated or deleted

- Identify access keys
  - aws iam generate-credential-report
  - aws iam get-credential-report --query 'Content' --output text | base64 –d
- Check if any access key is older than 90 days
- Create new access key and deactivate/delete the old one

2.16 Ensure IAM instance roles are used for AWS resource access from instances

- Using embedded credentials instead of IAM roles increases the risk of credential exposure and unauthorized access

- Check if the EC2 instance does not have an IAM role attached
  - aws ec2 describe-instances --region <region-name> --query 'Reservations[*].Instances[*].InstanceId'
  - aws ec2 describe-instances --region <region-name> --instance-id <Instance-ID> --query 'Reservations[*].Instances[*].IamInstanceProfile'

3.1.4 Ensure that S3 is configured with 'Block Public Access' enabled

- Prevents the accidental or malicious public exposure of data contained within the respective bucket

- Check 'Block Public Access' settings for every S3 bucket
  - aws s3 ls
  - aws s3api get-public-access-block --bucket <bucket-name>

3.2.3 Ensure that RDS instances are not publicly accessible

- Prevent that anyone and anything on the Internet can establish a connection to your database

- Check 'Publicly Accessible' flag on RDS is set to 'false'
  - aws rds describe-db-instances --region <region-name> --query 'DBInstances[*].DBInstanceIdentifier’
  - aws rds describe-db-instances --region us-east-1 --query 'DBInstances[*].[DBInstanceIdentifier,PubliclyAccessible]' --output table

3.2.1 Ensure that encryption-at-rest is enabled for RDS instances

- Data at rest protection: encrypt your data on the server that hosts your Amazon RDS DB instances

- Check 'StorageEncrypted' parameter value is set to 'true'
  - aws rds describe-db-instances --region –query 'DBInstances[*].[DBInstanceIdentifier,StorageEncrypted]' --output table
  - aws rds describe-db-instances --region <region-name> --db-instance-identifier <db-name> --query 'DBInstances[*].StorageEncrypted'

6.2 Ensure no Network ACLs allow ingress from 0.0.0.0/0 to remote server administration ports

- Prevent that no NACL allows unrestricted ingress access to remote server administration ports

- Check ACLs bound to different subnets
  - aws ec2 describe-network-acls --filters "Name=association.subnet-id,Values=*" --query 'NetworkAcls[].{ACL_ID:NetworkAclId, Subnet_ID:Associations[].SubnetId, Rules:Entries[?Egress==true && RuleAction==allow && (CidrBlock==0.0.0.0/0 || Ipv6CidrBlock==::/0)].{RuleNumber:RuleNumber, Action:RuleAction, Protocol:Protocol, CidrBlock:CidrBlock, Ipv6CidrBlock:Ipv6CidrBlock, PortRange:PortRange}}' –output table

6.3 Ensure no security groups allow ingress from 0.0.0.0/0 to remote server administration ports

- Prevent that no Security Group allows unrestricted ingress access to remote server administration ports

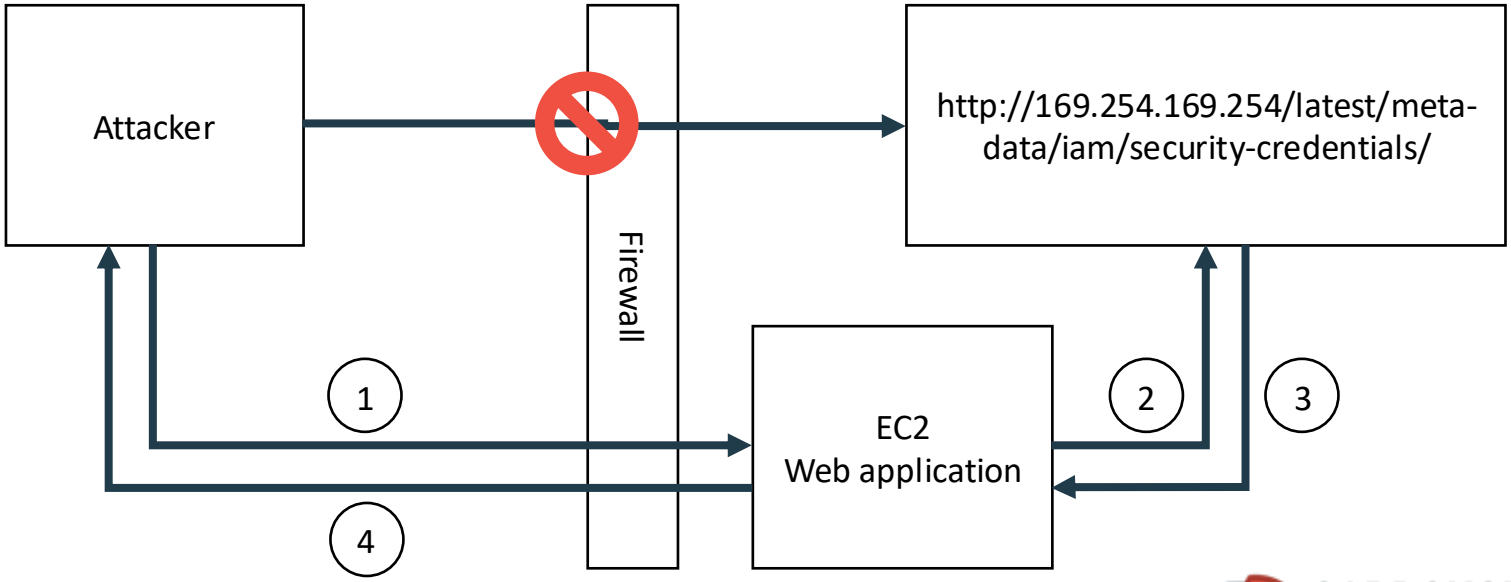
- Check all security groups for insecure inbound rules allowing traffic from 0.0.0.0/0
  - aws ec2 describe-security-group-rules --query 'SecurityGroupRules[?CidrIpv4 == "0.0.0.0/0" && IsEgress == `false`]' --output json

6.7 Ensure that the EC2 Metadata Service only allows IMDSv2

- Allowing Version 1 of the service may open EC2 instances to Server-Side Request Forgery (SSRF) attacks

- Determine whether the selected instance is using IMDSv2
  - aws ec2 describe-instances --region <region> --output table –query "Reservations[*].Instances[*].InstanceId"
  - aws ec2 describe-instances --region <region> --instance-ids <instance-id> --query "Reservations[*].Instances[*].MetadataOptions" --output table

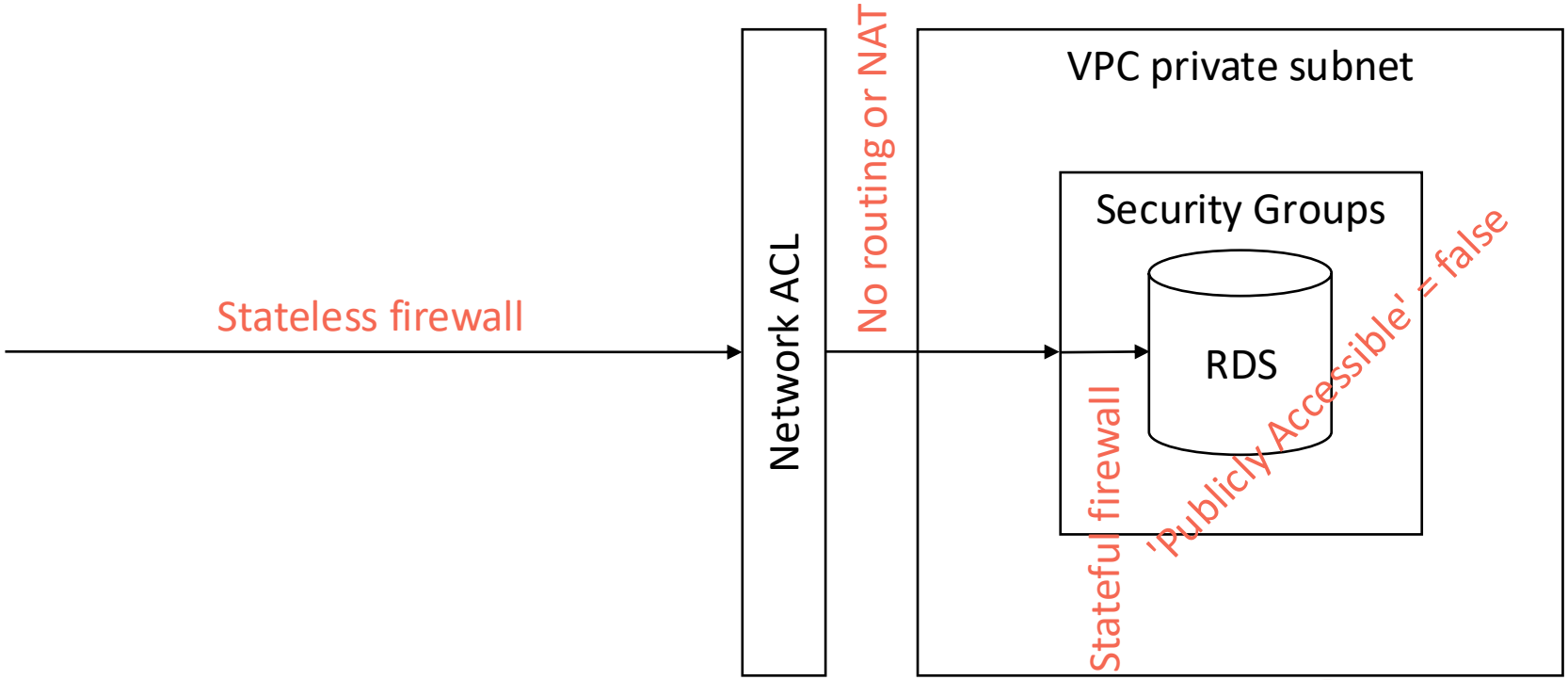
SSRF & metadata service



IMDSv2

- Two steps to query metadata service
  - Get temporary token
    - TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600"`
  - Query metadata service using the temporary token
    - curl -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-data/

Why can benchmark be useful?



Tools for config audit

CIS Foundations Benchmarks

Prescriptive guidance to securely configure public cloud accounts

Identity and Access Management

Guidance for passwords, user access, and MFA

Logging and Monitoring

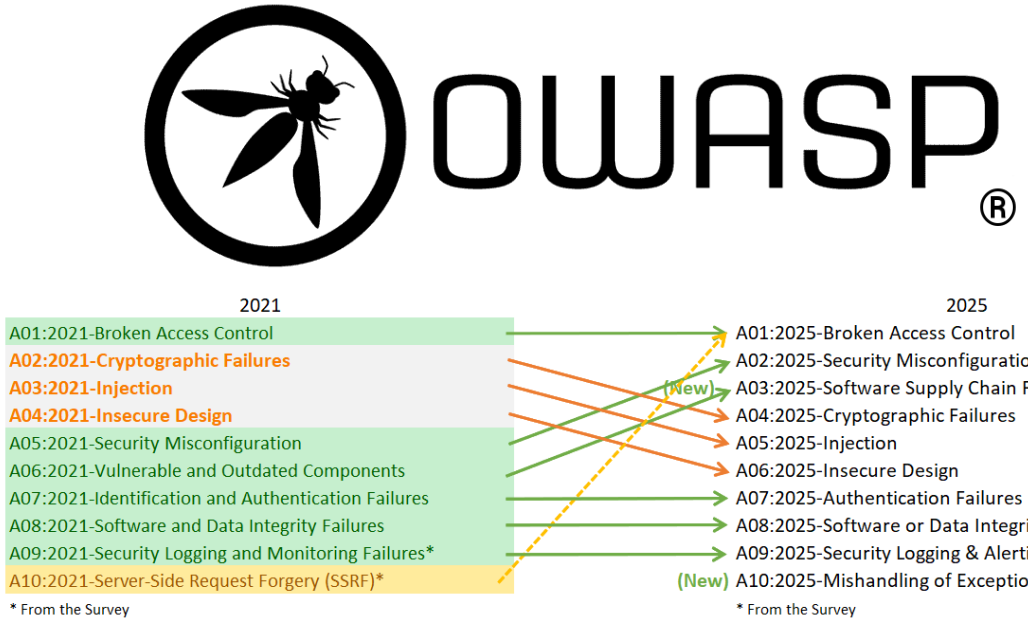
Steps to set up alerts, event tracking, and access policies

Networking

Best practices to restrict access and traffic

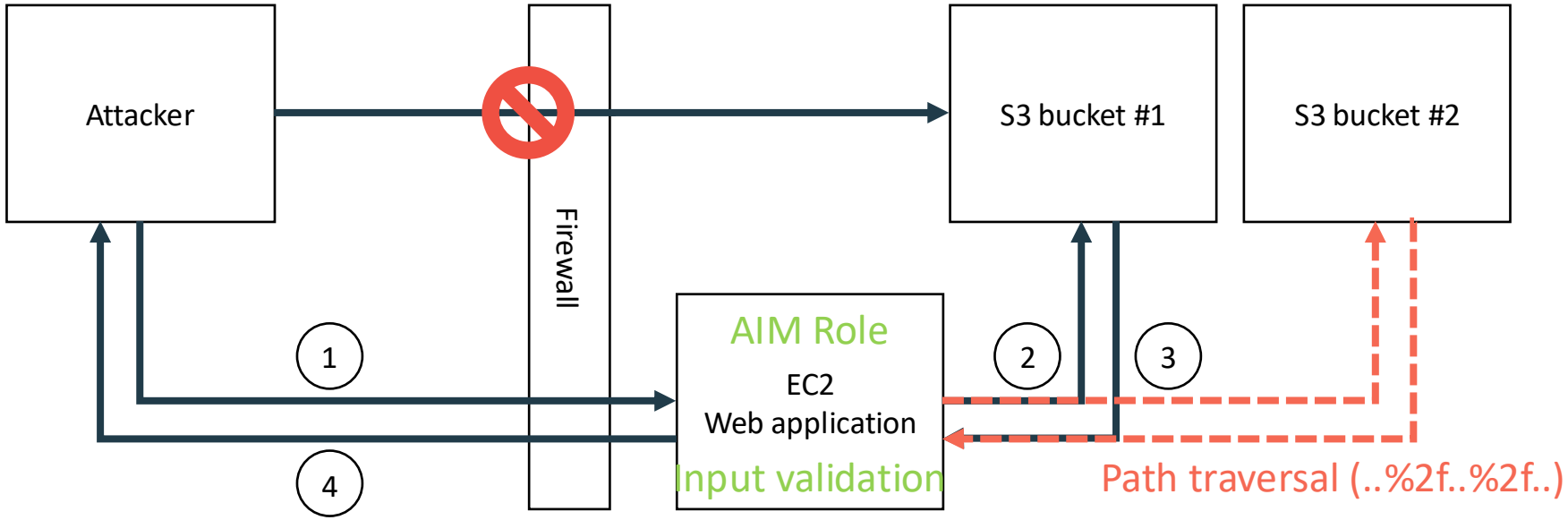
Many web apps exploits depend on input validation failure

- SQL injection
- XSS
- Directory traversal
- SSRF
- Etc.



CARBONSEC

SSRF & S3 bucket access



gpreseren@carbonsec.com