International Journal of Computer Applications Technology and Research
Volume 06–Issue 12, 562 - 566, 2017, ISSN:-2319–8656

Defense in Depth for Enterprise Public Cloud Workloads

Practical Security Patterns, Open-Source Assessment, and Continuous Compliance

Saurabh Srivastava
Independent Researcher
saurabhuser@gmail.com

Abstract: Enterprises moving workloads into public cloud are finding that the old security playbook doesn't transfer cleanly. The controls still matter, but now they have to work in an environment where infrastructure is built through APIs, workloads span dozens of accounts and VPCs, and configurations change several times a day. This article walks through security patterns that hold up in practice for enterprises running on AWS: account separation, network segmentation, identity and access management, centralized logging, configuration monitoring, and recurring assessment with open-source tools like Scout2, Prowler, Security Monkey, and Cloud Custodian. It also spends real time on what these tools cannot do. Service-to-service authentication, inspection of encrypted east-west traffic, container runtime monitoring, and governance across large account fleets are all still hard, and designs that pretend otherwise end up weaker than designs that admit it.

Keywords: component; formatting; style; styling; insert

1. INTRODUCTION

Public cloud [1-2] changes the mechanics of enterprise security more than the principles. Servers, storage, routes, access policies, and encryption settings are all created and modified through APIs[3-5]. That is what makes cloud fast. It is also what makes a bad template dangerous, because an overly broad permission doesn't stay in one place. The same automation that saves time will happily stamp the mistake across a hundred resources[6].

Defense in depth still applies. What changes is where the layers sit. In a data center, security teams leaned on a small number of network boundaries, central firewalls, long-lived servers, and manual change review. In AWS the boundary is smeared across the account, the VPC, the subnet, the security group, the IAM role, the key policy, the resource policy, and the deployment pipeline itself [7-15].

Any serious cloud security architecture has to answer a short list of questions. Which accounts and networks may talk to each other? Which identities can act on which resources? How is administrative activity recorded, and how quickly does an insecure change get noticed? What can be enforced automatically without breaking production? And how do you prove, months after deployment, that a control still works?

The rest of this article works through a practical model built from the capabilities available today.

2. PAGE SIZE SECURITY CHALLENGES IN ENTERPRISE CLOUD ADOPTION

2.1 The network perimeter is no longer enough

Internet-facing [16] traffic still matters, but it is a minority of what moves through a cloud environment. Applications talk across subnets, across VPCs, across accounts, and back to corporate data centers over private links [17-22].

Security groups [23] give you stateful filtering at the instance or interface level, and network ACLs add a subnet-level layer on top. Both are useful. Neither inspects application content, and neither tells you anything about the identity of the calling service. A firewall [24-27] at the internet edge protects one path out of many, so network controls have to be paired with workload permissions, application-level authentication, logging, and configuration monitoring.

2.2 Identity becomes part of the perimeter

IAM controls access to AWS APIs and resources. An IAM role hands an application temporary credentials, which means the application never has to store a permanent access key. Roles attach to EC2 instances and ECS tasks, so each workload can receive only the permissions its function requires.

That is a real improvement over access keys sitting in configuration files. It is not application-to-application identity, and the two get confused constantly. An ECS task role authorizes a container to reach an S3 bucket, a KMS key, or a DynamoDB table. It does nothing to authenticate a private HTTP call from one microservice to another. That problem has to be solved separately, with signed requests, client certificates, an identity-aware gateway, or whatever mechanism fits the stack.

2.3 Configuration changes create security drift

Cloud security lives and dies on configuration. A security group open to the wrong address range. An S3 bucket policy that permits public access. A CloudTrail trail someone quietly disabled. A wildcard buried in an IAM policy, an unencrypted volume, a route-table change that bypasses the inspection path, a key policy that lets far too many principals decrypt.

None of these are software vulnerabilities. Most arrive through ordinary administrative work: a copied template, a rushed change, a review that missed one line. And because resources change constantly, a quarterly assessment tells you almost nothing about the other eighty-nine days. Teams need configuration history, recurring evaluation, and an alert when something important moves.

2.4 Security information is scattered

No single source explains an incident. CloudTrail records API activity. AWS Config records what resources looked like and when they changed. VPC Flow Logs carry network metadata. Load balancers and web application firewalls see ingress, host agents see the operating system, applications hold the authentication and transaction context, and the identity provider knows who actually logged in.

Stitching an incident together takes centralized collection, consistent timestamps, and enough account and application context to correlate events in a SIEM. Without that enrichment, the logs are just expensive storage.

3. A LAYERED ENTERPRISE ARCHITECTURE

3.1 Separate workloads with AWS accounts

The AWS account is the strongest administrative boundary available, and it is underused. Separate accounts contain the blast radius of a configuration mistake, shrink the set of administrators who can touch sensitive workloads, and make cost and compliance reporting far less painful.

A workable structure separates production from nonproduction, regulated workloads from general ones, security and logging services from the applications they watch, and central network services from everything else. Give developers a sandbox account as well. The alternative is experimentation inside controlled environments, which is worse.

Cross-account access should go through IAM roles and temporary credentials, never shared users or copied keys. And account separation doesn't replace the controls inside each account. It gives you a boundary to apply them within.

3.2 Use dedicated connectivity carefully

Direct Connect provides private connectivity between enterprise facilities and AWS, usually with more predictable performance than the internet and support for hybrid routing designs. What it does not provide is encryption or compliance. I have heard "it goes over Direct Connect" offered as a security answer, and it isn't one. Route design, firewall enforcement, segmentation, encryption where the data warrants it, monitoring, redundant connectivity, and change control all still have to be engineered. Where confidentiality matters, encrypt above the link rather than trusting the link.

3.3 Segment virtual private clouds

Separate VPCs and subnets keep workloads with different trust levels apart, with security groups permitting only the protocols and sources each tier needs.

A common pattern is a dedicated VPC for centralized network or security services, and this is where VPC peering will disappoint you. Peering is not transitive, so two spoke VPCs cannot reach each other through a hub they both peer with. A true transit design needs routing or firewall appliances plus encrypted tunnels from every spoke, and the operational cost of that is easy to underestimate. Routing, availability, throughput, asymmetric paths, and failure recovery all take real engineering.

Central inspection is worth having where it earns its keep. Forcing every internal connection through a handful of appliances buys you a bottleneck and a single point of failure along with the visibility.

3.4 Protect internet-facing applications

Internet-facing web applications belong behind load balancing, content delivery, and, where the risk justifies it, a web application firewall. AWS WAF uses web access control lists and rules to allow or block requests reaching supported AWS resources; it is not a standalone reverse proxy. F5 Application Security Manager and Imperva SecureSphere are the common alternatives, and the right choice depends on the architecture and how deep inspection needs to go.

A WAF reduces exposure to common web attacks and abusive traffic patterns. It is not a substitute for secure development, authentication and authorization, input validation, dependency management, penetration testing, or rate controls in the application itself. Run new rules in monitoring mode first. A blocking rule with false positives interrupts real transactions, and the business will remember that longer than it remembers the attack you stopped.

3.5 Restrict AWS permissions

Design IAM policies around what a workload or administrator actually does. Prefer roles and temporary credentials over long-lived keys. Give applications with different jobs different roles. Avoid wildcards where practical, require multifactor authentication for privileged human access, keep cross-account trust narrow, and know exactly who can modify IAM policies or assume the powerful roles. CloudTrail should be recording all IAM and STS activity. Unused users, roles, and keys should be deleted rather than kept around just in case, and KMS key policies deserve the same scrutiny as the data they protect.

Perfect least privilege on the first deployment is a fantasy. What works in practice: start with a restricted functional policy, watch which API calls the application actually makes, and tighten from there.

3.6 Keep secrets out of source code

Access keys, database passwords, and private tokens do not belong in repositories. IAM roles remove much of the need. For the rest, pre-commit scanning, CI checks, restricted repository access, rotation, and environment-specific secret delivery each remove a slice of the risk. git-secrets catches strings that look like AWS access keys before they are committed. Entropy-based scanners find more but need tuning, because randomly generated identifiers look a lot like credentials.

Treat scanning as a safety net, not a secret-management system. It cannot recognize every custom credential format, and it cannot tell you whether a detected value is still live. When a key does leak, revoke first and investigate second.

4. LOGGING AND DETECTION

4.1 Record administrative activity

Turn CloudTrail on in every account and region in use, and deliver the logs somewhere workload administrators cannot touch them. The design questions are mundane, but they decide whether the logs survive an incident: protection from deletion, encryption, defined retention, alerts when someone modifies a trail, time synchronization, and a clear answer to who can read the logs at 2 a.m. during an investigation. Keep

the log custodians separate from the people whose actions the logs record.

CloudTrail covers AWS API activity. Operating system, application, and network monitoring are separate problems and stay that way.

4.2 Collect network-flow information

VPC Flow Logs record metadata about accepted and rejected flows, which is enough to investigate unexpected connections, chase denied traffic, and see how application tiers actually talk to each other. They capture no packet contents and carry no business context. A flow record becomes useful when you can enrich it: which account, which VPC and subnet, which application, whose environment, what classification.

In a large estate, think before shipping every record to a premium SIEM index. The bill arrives faster than the insight.

4.3 Preserve application context

Infrastructure logs can prove that one address connected to another. Usually only the application knows which user, which transaction, and whether the request was authorized. Applications should emit structured events for authentication successes and failures, privilege changes, sensitive data access, administrative actions, rejected requests, significant transaction failures, and security-control failures.

Shared request identifiers matter more than any single log source. They are what let you follow one transaction across the load balancer, the services, the database, and the cloud API.

5. Configuration monitoring and compliance

5.1 AWS Config

AWS Config records supported resource configurations and their changes over time, and Config Rules evaluate resources against defined requirements. The managed rules catch things like S3 buckets open to public read or write and can notify operators when permissions change. Rules also exist for required logging, encryption settings, approved configurations, tagging, security-group restrictions, and CloudTrail state.

Config has limits, though. Coverage depends on which resource types Config supports, and custom requirements mean Lambda-backed rules that someone has to write, test, and maintain.

5.2 Scout2

Scout2 scans an AWS account and produces a report across services including IAM, EC2, S3, RDS, CloudTrail, and VPC. It is widely used for exactly what it is good at: periodic review and catching common configuration weaknesses.

The report is a snapshot. It says nothing about the account five minutes after the scan finished, and it should never be filed as proof of ongoing compliance.

5.3 Prowler

Prowler runs checks based largely on the CIS AWS Foundations Benchmark, which makes it a natural fit for initial account reviews, recurring benchmark runs, validation after major changes, and audit evidence. It is also a quick way to find missing logging and monitoring controls.

Remediation should stay a separate, controlled step. A finding needs a human who understands the workload before it becomes a change.

5.4 Security Monkey

Security Monkey, from Netflix, watches cloud-account configurations and flags changes and risky settings, with useful visibility across many accounts. The visibility is real. So is the cost: it brings its own infrastructure, database, permission set, patching, and availability requirements. Decide whether the visibility pays for that operating burden before adopting it.

5.5 Cloud Custodian

Cloud Custodian expresses governance policies in YAML and applies them to AWS resources. Organizations are running it across dozens of accounts for notification, tagging, and selected enforcement: flagging over-permissive security groups, finding untagged or unencrypted resources, notifying owners, stopping unauthorized instances, reacting to specific CloudTrail events.

Enforcement is where teams hurt themselves. A policy that is technically correct but badly scoped will act on a great many resources very quickly, and Custodian will not ask whether you meant it. Roll out in stages: test against representative nonproduction resources, run report-only, validate findings with application owners, enforce on a narrow scope, log every automated action, keep a rollback and exception path, and only then widen. Skipping steps here is how automation turns a small mistake into a large one.

6. INTEGRATION INTO THE DELIVERY PIPELINE

Security checks belong as close as possible to the point where changes enter. A delivery pipeline can run static analysis on infrastructure templates, secret scanning, dependency checks, container image scanning, Prowler or custom account checks, tag validation, verification that logging and encryption are enabled, and review of security-group changes before anything ships.

Not every security decision reduces to pass or fail. The pipeline can tell you a security group opens a port to the internet. It cannot tell you whether the application design justifies that. The strongest pipeline controls are specific, testable, and tied to an approved standard. Ambiguous findings should go to a person for review, not into an automatic block that teaches developers to route around the pipeline.

7. LIMITATIONS OF THE CURRENT CONTROL MODEL

7.1 Service-to-service identity is still hard

IAM authorizes calls to AWS services well. It does not authenticate an enterprise’s own services to each other, and there is still no consistent answer for that. Istio, publicly introduced in May 2017, demonstrates service identity, access control, and encrypted service communication in its early releases, and the early releases look promising. It is not yet

something to bet a production environment on. For now, service authentication means application-specific tokens, gateways, certificates, or whatever established mechanism the stack already supports.

7.2 Encrypted east-west traffic resists inspection

Network firewalls can classify standard protocols and read connection metadata, but custom applications talking over TLS are opaque without decryption. And decryption drags in certificate and key management, application compatibility, privacy questions, performance cost, failure handling, and the awkward matter of who gets to see the cleartext. Routing HTTPS through a firewall does not, by itself, produce application-level visibility, whatever the datasheet implies.

7.3 Container security is still maturing

Container security is a set of problems rather than a product category: image provenance, vulnerability scanning, host hardening, runtime privileges, Linux capabilities, secrets, network access, logging, and patch processes. No single tool covers that list today. The practical combination is image scanning plus host controls plus restricted task definitions plus monitoring, with rapid replacement of vulnerable images standing in for traditional patching.

7.4 Multi-account governance takes engineering

The tools find insecure configurations. The organization still has to decide who owns each finding, how severe it is, who fixes it, by when, and what evidence gets kept. Policy ownership, account onboarding, exception approval, escalation, measurement, tool maintenance. None of it is glamorous, and all of it determines whether findings get fixed or pile up in a dashboard nobody reads.

In my experience the hard part was never detection. It is getting a straight answer to two questions: whose problem is this, and does it actually matter?

8. A recommended enterprise baseline

A defensible minimum for an enterprise AWS environment:

-Separate production, nonproduction, regulated, logging, and shared-service environments where the risk justifies it.

-Use IAM roles and temporary credentials instead of stored access keys.

-Enable CloudTrail across every region in use and protect the destination logs.

-Use AWS Config for configuration history and the compliance rules it supports.

-Restrict inbound and outbound paths through security groups, network controls, and documented routing.

-Put internet-facing applications behind a web application firewall where the risk warrants it.

-Centralize security-relevant logs and enrich them with account, environment, and application context.

-Run recurring assessments with Prowler or Scout2.

-Introduce Cloud Custodian or similar automation in reporting mode first.

-Scan repositories for credentials and revoke exposed keys immediately.

-Define ownership, exceptions, remediation expectations, and rollback procedures.

-Test controls for availability and failure behavior, not only enforcement.

9. CONCLUSION

Cloud security is not perimeter security relocated into someone else's data center. The controls distribute across accounts, identities, networks, resource policies, pipelines, logs, and monitoring, and each layer covers for the failures of the others. Account separation limits blast radius. Roles remove standing credentials. CloudTrail, Config, and Flow Logs each record a different kind of evidence. Scout2 and Prowler make assessment recurring instead of annual, and Security Monkey and Cloud Custodian add change detection and enforcement where they justify their upkeep.

The gaps matter just as much. IAM does not authenticate service-to-service traffic. VPC peering is not transitive. Firewalls cannot see inside encrypted custom protocols. Automated remediation amplifies mistakes exactly as efficiently as it corrects them.

Nobody gets perfect security. What an enterprise can get is a set of layers it understands, can test, can recover when they fail, and can keep current with infrastructure that changes every day. That is as much operational discipline as control selection, and the operational half is usually where the work is.

3. REFERENCES

[1] Gartner. Forecast: Public Cloud Services, Worldwide, 2014-2020, 2Q16 Update. Gartner Research, 2016.

[2] Right Scale. State of the Cloud Report 2017. RightScale Inc., 2017.

[3] Cloud Security Alliance. Security Guidance for Critical Areas of Focus in Cloud Computing, Version 4.0. CSA, 2017.

[4] N. MacDonald, P. Firstbrook. Market Guide for Cloud Workload Protection Platforms. Gartner Research, 2017.

[5] J. Kindervag. No More Chewy Centers: Introducing the Zero Trust Model of Information Security. Forrester Research, 2010.

[6] R. Ward, B. Beyer. BeyondCorp: A New Approach to Enterprise Security. USENIX ;login:, Vol. 39, No. 6, December 2014.

[7] B. Osborn, J. McWilliams, B. Beyer, M. Saltonstall. BeyondCorp: Design to Deployment at Google. USENIX ;login:, Vol. 41, No. 1, Spring 2016.

[8] L. Cittadini, B. Spear, B. Beyer, M. Saltonstall. BeyondCorp: The Access Proxy. USENIX ;login:, Vol. 41, No. 4, Winter 2016.

[9] Amazon Web Services. AWS Direct Connect User Guide. docs.aws.amazon.com/directconnect/, accessed 2017.

[11] Palo Alto Networks. VM-Series Deployment Guide for Amazon Web Services. Palo Alto Networks Technical Documentation, 2016-2017.

[12] Amazon Web Services. Transit VPC Solution. AWS Solutions Library, 2016-2017.

[13] Imperva. SecureSphere Web Application Firewall Administration Guide. Imperva Product Documentation, 2017.

[14] SPIFFE Project. SPIFFE: Secure Production Identity Framework for Everyone. github.com/spiffe, accessed 2017.

[15] Evident.io. Evident Security Platform Product Documentation. 2017. (Palo Alto Networks acquired Evident.io, March 2017.)

[16] Redlock. Cloud Security and Compliance Product Documentation. 2017.

[17] L. Simon et al. Scout2 - Security Auditing Tool for AWS Environments. NCC Group. github.com/nccgroup/Scout2, accessed 2017.

[18] T. de la Fuente. Prowler - AWS CIS Benchmark Tool. github.com/toniblyx/prowler, accessed 2017.

[19] Center for Internet Security. CIS Amazon Web Services Foundations Benchmark, Version 1.1.0. November 2016.

[20] Capital One. Cloud Custodian - Rules Engine for Cloud Security, Cost and Governance. github.com/capitalone/cloud-custodian, accessed 2017.

[21] Netflix. Security Monkey. github.com/Netflix/security_monkey, accessed 2017.

[22] Amazon Web Services Labs. git-secrets - Prevents Committing Secrets and Credentials into Git Repositories. github.com/awslabs/git-secrets, accessed 2017.

[23] D. Ayrey. truffleHog - Searches Through Git Repositories for High Entropy Strings and Secrets. github.com/dxa4481/truffleHog, accessed 2017.

[24] Amazon Web Services. AWS Security Best Practices. Whitepaper, 2016.

[25] Amazon Web Services. AWS Well-Architected Framework - Security Pillar. Whitepaper, 2016-2017.

[26] VMware. Micro-segmentation with VMware NSX. VMware Technical Documentation, 2016.

[27] Illumio. Adaptive Security Platform: Micro-Segmentation for Data Centers and Clouds. Illumio Product Documentation, 2016-2017.