A Penetration Tester's Guide to Exploiting AWS Elastic Beanstalk

Anatomy of an AWS Elastic Beanstalk Environment: An Attacker's Perspective

AWS Elastic Beanstalk is a Platform-as-a-Service (PaaS) offering designed to abstract the complexities of infrastructure management, allowing developers to deploy applications by simply uploading their code.[1] While this abstraction provides significant operational benefits, it also creates a standardized and predictable environment that can be systematically deconstructed and exploited by a knowledgeable attacker. Understanding the anatomy of this service from an offensive security standpoint is the first step toward identifying and leveraging its weaknesses. An Elastic Beanstalk deployment is not a monolithic entity but a collection of interconnected AWS services, each representing a potential point of failure or an avenue for exploitation.[3]

Deconstructing the Architecture: From CNAME to Shell

The typical lifecycle of a request to an Elastic Beanstalk application follows a well-defined path, exposing several key components that a penetration tester must scrutinize. This predictable data flow provides multiple opportunities for reconnaissance, interception, and exploitation.[5]

The Data Flow

A standard request begins with a DNS resolution. Every Elastic Beanstalk environment is assigned a unique CNAME (Canonical Name) record, which typically follows the format {environment-name}.{random-string}.{region}.elasticbeanstalk.com.[5] This URL is an alias that points to an Elastic Load Balancer (ELB), which serves as the public-facing entry point for all incoming traffic.[5] The ELB, in turn, distributes requests across a fleet of Amazon Elastic Compute Cloud (EC2) instances managed by an Auto Scaling group. These EC2 instances are where the application code actually runs.[5] This entire collection of resources—the application, its versions, and the running infrastructure—constitutes an Elastic Beanstalk environment.[4]

Core Components as Targets

From an attacker's perspective, each component in this architecture is not just a functional unit but a potential target:

- **Application & Application Version**: In Elastic Beanstalk terminology, an "Application" is a logical container for its various components, including different "Application Versions".[4] For a penetration tester, the Application Version is of paramount importance. It represents a specific iteration of deployable code, and crucially, it corresponds to a source bundle (e.g., a ZIP or WAR file) stored as an object in an Amazon Simple Storage Service (S3) bucket.[7] Gaining access to this S3 object means gaining access to the application's complete source code.
- **Environment & Environment Tier**: An "Environment" is the live, running instance of an Application Version.[7] Environments are categorized into tiers, primarily the "Web Server" tier for handling HTTP requests and the "Worker" tier for processing background tasks.[7] While Web Server tiers are the most common initial targets due to their public exposure, a compromised Worker tier can be more valuable. Worker environments are designed to interact with backend services like Amazon Simple Queue Service (SQS) and Amazon DynamoDB, and their associated IAM roles often possess more extensive permissions to facilitate these interactions, offering a richer post-exploitation landscape.[7]
- **IAM Instance Profile (aws-elasticbeanstalk-ec2-role)**: This is arguably the most critical component from an exploitation standpoint. Each EC2 instance within an Elastic Beanstalk environment is launched with an attached IAM role, contained within an instance profile.[11] This role, often the default aws-elasticbeanstalk-ec2-role, grants the instance temporary security credentials to interact with other AWS services on behalf of the application.[9] If an attacker can compromise an EC2 instance (e.g., through a web application vulnerability), they can steal these credentials from the instance metadata service, providing them with a foothold in the AWS control plane.
- **S3 Source Bucket (elasticbeanstalk-region-account-id)**: When an application is deployed, Elastic Beanstalk creates a dedicated S3 bucket to store application source bundles and, optionally, server logs.[8] This bucket follows a predictable naming convention: elasticbeanstalk-{region}-{account-id}.[9] The default permissions granted by the aws-elasticbeanstalk-ec2-role often allow the EC2 instances to read from and write to this bucket, creating a powerful vector for source code theft and tampering attacks.[9]

Platform Fingerprinting: Exploiting Predictability

The core value proposition of a PaaS like Elastic Beanstalk—abstracting away infrastructure complexity—directly translates into a strategic advantage for an attacker. This abstraction enforces standardization across deployments. When a developer selects a platform, such as "Python 3.12 running on 64bit Amazon Linux 2023," Elastic Beanstalk provisions a known Amazon Machine Image (AMI) with a pre-configured software stack.[13] This creates a highly predictable environment, from the file system layout and log locations to the specific reverse proxy configuration. An attacker who gains shell access does not need to perform extensive internal reconnaissance to understand the server's architecture; they can operate with a pre-existing "map" of the target system. This predictability significantly reduces the time from initial compromise to lateral movement, as the developer's convenience becomes the attacker's operational blueprint.[1]

Identifying the specific platform in use is a crucial first step, as it provides a clear roadmap for subsequent exploitation. Each platform has distinct signatures:

- **Docker**: The presence of a Dockerrun.aws.json file in the source code is a clear indicator of a Docker-based deployment. Internally, the instance profile will likely have the AWSElasticBeanstalkMulticontainerDocker managed policy attached, which grants permissions to interact with Amazon Elastic Container Service (ECS) to coordinate container deployments.[16]
- **Go**: Go applications on Elastic Beanstalk are often deployed either as a source file named application.go or as a pre-compiled binary named application. By default, the platform configures the reverse proxy to forward traffic to the Go application running on port 5000.[19]
- **Java (Tomcat)**: These environments are characterized by the deployment of Web Application Archive (.war) files. Fingerprinting can involve probing for default Tomcat error pages or management interfaces that may have been inadvertently exposed. The platform includes a reverse proxy that can be configured to serve static assets, a configuration that can be enumerated post-compromise.[21]
- **Node.js**: The Node.js platform typically uses NGINX as its default reverse proxy. The application's dependencies are defined in a package.json file. On Amazon Linux 2, application logs, including standard output, are commonly found at /var/log/web.stdout.log, providing a predictable location to search for sensitive information leakage.[23]
- **Python**: Python environments are identified by the use of a requirements.txt file for managing dependencies and a specific WSGI (Web Server Gateway Interface) path configuration that tells the web server where the application entry point is. Environment variables, which are often a source of credential leakage, are sourced from a file located at /opt/python/current/env on the instance.[25]

Reconnaissance and Target Identification

Effective reconnaissance is the foundation of any successful penetration test. For Elastic Beanstalk, this process can be divided into two distinct phases: external footprinting, which involves identifying targets from a black-box perspective without any prior access, and internal enumeration, which leverages some form of initial access (typically a set of AWS credentials) to map the environment with high fidelity.

External Footprinting: Finding Beanstalk in the Wild

Discovering Elastic Beanstalk environments from the public internet relies on identifying the underlying infrastructure components that are, by necessity, publicly exposed.

- **IP Range Scanning**: The most prevalent and effective method for discovering public-facing web applications, including those on Elastic Beanstalk, is the continuous scanning of known public cloud IP address ranges.[6] Threat actors and security researchers constantly probe these ranges for open ports and services. This means that a newly deployed, publicly accessible Elastic Beanstalk environment can be discovered and scanned for common vulnerabilities within minutes of its creation, regardless of whether its DNS name has been shared.[28] This reality renders security through obscurity ineffective.
- **DNS Enumeration**: While the default Elastic Beanstalk URL contains a randomized string to prevent trivial enumeration, many organizations assign custom domain names to their environments using a CNAME record that points to the Beanstalk URL.[5] These custom domains are discoverable through standard DNS reconnaissance techniques, such as querying certificate transparency logs, using DNS brute-forcing tools, or analyzing public DNS datasets.
- **HTTP Header Fingerprinting**: The response headers from a web server can provide clues about the underlying infrastructure. Elastic Beanstalk environments fronted by an Application Load Balancer often return a Server: awselb/2.0 header, although this can be disabled.[29] Other headers, such as X-Forwarded-For, X-Forwarded-Proto, and X-Forwarded-Port, are added by Classic Load Balancers to provide the backend application with information about the original client request.[31] The presence of these headers strongly suggests that the application is behind an AWS load balancer, a key component of a standard Elastic Beanstalk setup.

Internal Enumeration: Leveraging Initial Access

The transition from external to internal reconnaissance represents an exponential leap in an attacker's capabilities. While external scanning relies on inference and can be noisy, the acquisition of even a single, low-privilege AWS access key provides direct, authoritative access to the AWS API. This fundamentally shifts the nature of the engagement from a blind test to a fully mapped internal assessment. The API provides structured, ground-truth data, laying bare the entire attack surface of the target environment. Consequently, the primary objective of the initial phase of any AWS penetration test should be the acquisition of valid credentials.

- **The Power of AWS CLI**: Once an attacker possesses AWS credentials, the AWS Command Line Interface (CLI) becomes their most powerful reconnaissance tool. The foundational command for enumerating Elastic Beanstalk is aws elasticbeanstalk describe-environments.[1] Even with highly restricted permissions, this command can reveal a wealth of information, including precise environment names, application names, the platform ARN (which specifies the exact software stack), and the environment's current status.[1] This information is crucial for planning subsequent attack steps.

- **Automating with Pacu**: Pacu, an open-source AWS exploitation framework often described as the "Metasploit for AWS," provides modules to automate and streamline this process.[1] The elasticbeanstalk__enum module is specifically designed for this purpose. With a single command, it can enumerate all applications, environments, configurations, and tags associated with the provided credentials. Critically, it goes a step further by automatically parsing the environment configurations to identify and flag potential secrets stored in environment variables, and it can optionally download application source code bundles from S3 for offline analysis.[15]
- **Enumerating Platform Versions**: To assess the risk of vulnerabilities in the underlying platform, it is essential to identify the specific version in use. The aws elasticbeanstalk list-platform-versions command allows a tester to list all platform versions available to the account.[34] By correlating this information with the PlatformArn from the describe-environments output, the tester can pinpoint the exact software stack. This is the first step in researching known Common Vulnerabilities and Exposures (CVEs) that may affect the environment's operating system, runtime, or web server.[36]

Initial Compromise: Common Attack Vectors and Exploitation Techniques

Gaining an initial foothold in an AWS Elastic Beanstalk environment can be achieved through several distinct attack vectors. These range from exploiting common cloud misconfigurations to leveraging traditional web application vulnerabilities. A key theme across these vectors is their convergence on a single, critical objective: obtaining the IAM credentials associated with the underlying EC2 instance profile. These credentials serve as the bridge from the compromised data plane (the application and its server) to the AWS control plane (the API), enabling post-exploitation activities and lateral movement.

Vector 1: Leaked Credentials in Environment Variables

One of the most common and high-impact misconfigurations is the storage of sensitive information, such as database credentials, API keys, or even secondary AWS access keys, directly within Elastic Beanstalk environment variables.[1] Developers often use environment variables for their convenience, but this practice turns the environment's configuration into a treasure trove for an attacker with even minimal read access.

- **Discovery**: An attacker with permissions to call elasticbeanstalk:DescribeConfigurationSettings or elasticbeanstalk:DescribeEnvironments can directly retrieve all environment variables in plaintext.[1] Tools like Pacu, with its
elasticbeanstalk__enum module, automate this discovery process, highlighting any variables that appear to

contain secrets.[33]

- **Walkthrough: The CloudGoat beanstalk_secrets Scenario**: This scenario provides a practical, step-by-step demonstration of how leaked environment variables can lead to a full account compromise.[32]
  1. **Initial Access**: The attacker starts with a set of low-privileged AWS access keys that have limited permissions, primarily related to describing Elastic Beanstalk resources.[32]
  2. **Enumeration**: Using these initial keys, the attacker runs Pacu's elasticbeanstalk__enum module. The module queries the environment's configuration and discovers two critical environment variables: SECONDARY_ACCESS_KEY and SECONDARY_SECRET_KEY.[32]
  3. **Credential Pivoting**: The attacker configures a new AWS CLI profile using these newly discovered secondary credentials.
  4. **Permission Enumeration and Privilege Escalation**: With the secondary user's context, the attacker uses Pacu's iam__enum_permissions and iam__privesc_scan modules. The scan reveals that this secondary user has the iam:CreateAccessKey permission, which allows them to create new access keys for other IAM users.[32]
  5. **Exploitation**: The attacker targets a known administrative user and uses the iam:CreateAccessKey permission to generate a new set of access keys for that admin user.
  6. **Full Compromise**: The attacker configures a third AWS CLI profile with the newly created administrator keys. With these credentials, they have full control over the AWS account and can achieve their final objective, such as retrieving a flag stored in AWS Secrets Manager.[32]

Vector 2: Server-Side Request Forgery (SSRF) and the Instance Metadata Service (IMDS)

A Server-Side Request Forgery (SSRF) vulnerability in a web application hosted on Elastic Beanstalk can be escalated into a critical security breach. This cloud-native attack chain allows an attacker to force the server-side application to make requests to internal network resources, with the primary target being the EC2 Instance Metadata Service (IMDS).[39]

- **IMDSv1 vs. IMDSv2**: The vulnerability of this service largely depends on the version in use. IMDSv1 is a simple request-response service accessible via a non-routable IP address (169.254.169.254) from within the EC2 instance.[41] It requires no session token or special headers, making it trivial to query with a basic GET request spawned by an SSRF vulnerability.[42] In contrast, IMDSv2 is session-oriented. To access metadata, a client must first obtain a session token via a PUT request and include that token in a header for all subsequent GET requests. This session-oriented approach effectively mitigates most common SSRF attacks, as they are typically limited to crafting GET requests.[41] Despite the enhanced security of IMDSv2, its adoption has been slow; research from 2022 indicated that as many as 93% of EC2 instances did not enforce its use, making IMDSv1 exploitation a highly relevant attack vector.[41]
- **Exploitation Walkthrough**:
  1. **Identify SSRF**: The attacker first identifies an SSRF vulnerability within the web application, such as a

parameter that accepts a URL to fetch remote content (e.g., an image importer or a webhook processor).[9]

2. **Target IMDS**: The attacker crafts a payload that directs the vulnerable application to make a request to the IMDS endpoint. The initial probe would target the root of the metadata service: http://169.254.169.254/latest/meta-data/.[39]
3. **Enumerate IAM Role**: By traversing the metadata directory structure, the attacker queries the /iam/security-credentials/ path. The response to this query will be the name of the IAM role attached to the instance, which in an Elastic Beanstalk environment is often the default aws-elasticbeanstalk-ec2-role.[9]
4. **Steal Credentials**: With the role name identified, the attacker makes a final request to http://169.254.169.24/latest/meta-data/iam/security-credentials/{ROLE_NAME}. The IMDS responds with a JSON object containing the temporary AccessKeyId, SecretAccessKey, and Token for the instance profile.[39]
5. **Gain Control Plane Access**: The attacker configures their local AWS CLI with these stolen temporary credentials, granting them the ability to execute AWS API calls with the same permissions as the compromised EC2 instance.

Vector 3: Exploiting Outdated and Vulnerable Platform Versions

Elastic Beanstalk platforms, which consist of an operating system, runtime, application server, and web server, are regularly updated by AWS. However, organizations may fail to apply these updates, leaving their environments running on "deprecated" or "retired" platform versions that no longer receive security patches.[45] This creates a significant attack surface, as these platforms may contain publicly known and exploitable vulnerabilities.

- **Identification**: A penetration tester can identify the platform version using the PlatformArn field in the output of the aws elasticbeanstalk describe-environments command. This ARN can then be cross-referenced with AWS documentation to determine if the platform is retired.[45]
- **Exploitation Strategy**:
  1. **Component Analysis**: Once an outdated platform is identified, the tester must determine the specific versions of its components (e.g., Amazon Linux 1, NGINX 1.18, Python 3.6). Sometimes, a platform for one language may even include outdated packages for another, such as an old version of Ruby being included in a PHP environment, expanding the attack surface.[46]
  2. **Vulnerability Research**: The tester then searches public vulnerability databases, such as the National Vulnerability Database (NVD) or the Amazon Linux Security Center, for CVEs affecting these specific component versions.[47]
  3. **Exploitation**: If a viable exploit for a vulnerability like remote code execution (RCE) is available, the tester can deploy it against the environment's public endpoint to gain an initial shell on the underlying EC2 instance. From there, they can proceed to steal credentials from the IMDS.

Vector 4: Direct Application-Layer Exploitation (RCE)

Beyond cloud-specific misconfigurations, the application code deployed on Elastic Beanstalk remains a primary attack surface. Standard web application vulnerabilities, if present, can provide a direct path to compromising the underlying server.[48]

- **Vulnerability Types**: Penetration testers should probe for common high-impact vulnerabilities such as:
  - **Remote Code Execution (RCE)**: Flaws that allow an attacker to execute arbitrary code or commands on the server, often through insecure file uploads, unsafe deserialization, or command injection.[49]
  - **SQL Injection**: Vulnerabilities that allow an attacker to interfere with the queries that an application makes to its database, potentially leading to data exfiltration or, in some database configurations, command execution.
- **Exploitation**: If an attacker successfully exploits an RCE vulnerability, they gain a shell on the EC2 instance. This outcome is functionally identical to a successful platform CVE exploit. The attacker's immediate next step is to query the IMDS endpoint (http://169.254.169.254) to retrieve the instance profile's IAM credentials, thereby gaining access to the AWS control plane.[48]

Post-Exploitation: Abusing Service Integrations and Achieving Persistence

Once an initial foothold is established within an Elastic Beanstalk environment—typically by acquiring the IAM credentials from the EC2 instance profile—the focus of an attack shifts to post-exploitation. This phase involves leveraging the compromised role's permissions to pivot deeper into the AWS account, exfiltrate data, achieve privilege escalation, and establish long-term persistence. The standardized architecture of Elastic Beanstalk and its tight integration with other AWS services, particularly S3, provide fertile ground for these activities.

A critical realization for any penetration tester is that the very mechanisms designed for automated deployment and scaling in Elastic Beanstalk can be weaponized. The deployment process itself can be subverted to function as a reliable code execution engine for an attacker. By gaining write access to the S3 bucket where application source code is stored, an attacker can plant a payload. They can then trigger a redeployment through the API or simply wait for a natural auto-scaling event, which causes new instances to automatically pull and execute the malicious code. In this model, the S3 bucket becomes the payload repository, and the Elastic Beanstalk service API and auto-scaling functionality become the remote execution trigger, allowing an attacker to achieve their objectives using the intended functionality of the service.

Pivoting Through the S3 Source Bucket: The Code Tampering Attack

One of the most potent post-exploitation techniques involves abusing the default permissions granted to the Elastic Beanstalk instance profile to tamper with the application's source code.

- **Analyzing the AWSElasticBeanstalkWebTier Policy**: The default aws-elasticbeanstalk-ec2-role is often attached with the AWSElasticBeanstalkWebTier managed policy. A close examination of this policy reveals that it grants permissions such as s3:Get*, s3:List*, and, most critically, s3:PutObject on S3 buckets with names matching the pattern elasticbeanstalk-*.[9] This provides a direct path to modify the application's core logic.
- **Exploitation Walkthrough**:
  1. **Identify and Access the Bucket**: Using the stolen instance profile credentials, the attacker first identifies the target S3 bucket, which follows the predictable naming convention elasticbeanstalk-{region}-{account-id}.[9]
  2. **Download Source Code**: The attacker lists the contents of the bucket to locate the source code bundles for the various application versions. They then download the bundle for the currently deployed version to their local machine.[9]
  3. **Inject Backdoor**: The attacker unzips the source bundle, injects a backdoor (e.g., a simple web shell in PHP, Python, or the application's native language), and then re-packages the bundle.[9]
  4. **Overwrite Source Code**: Leveraging the s3:PutObject permission, the attacker uploads the malicious source bundle back to the S3 bucket, overwriting the legitimate application code.[9]
  5. **Trigger Redeployment**: To make the backdoor live, the modified code must be deployed to the running EC2 instances. An attacker can trigger this in several ways:
     - **API-driven Redeployment**: If the compromised role has permissions like elasticbeanstalk:RebuildEnvironment or elasticbeanstalk:UpdateEnvironment, the attacker can directly invoke the API to force a redeployment of the now-malicious source code.[9]
     - **CI/CD Pipeline Trigger**: If the environment uses a CI/CD service like AWS CodePipeline that is configured to trigger deployments upon changes to the S3 source bucket, the attacker's upload will automatically initiate the deployment process.[9]
     - **Passive Triggering**: The attacker can also wait for a legitimate developer to push an update or for an auto-scaling event to occur. When a new EC2 instance is launched, it will pull the latest application version from S3, which now contains the backdoor.
  6. **Persistent Access**: Once the malicious code is deployed, the attacker can access their web shell via its URL, providing them with persistent remote code execution capabilities on the environment's instances.

Lateral Movement from a Compromised Beanstalk Instance

A compromised Elastic Beanstalk instance serves as an excellent pivot point for moving laterally within the target's cloud environment.

- **Network Pivoting**: Since the EC2 instance resides within a Virtual Private Cloud (VPC), it can be used as a beachhead to scan the internal network. This allows the attacker to discover and probe other resources, such as databases, caching servers, or internal administrative panels, that are not directly exposed to the public internet but are accessible from within the VPC.[53]
- **Control Plane Pivoting**: The stolen IAM credentials are the key to moving laterally across the AWS control plane. The default AWSElasticBeanstalkWebTier and AWSElasticBeanstalkWorkerTier policies grant access to services beyond S3, including CloudWatch Logs, AWS X-Ray, SQS, and DynamoDB.[52] An attacker can query these services to exfiltrate sensitive application data, logs containing credentials, or discover further misconfigurations.
- **Advanced Lateral Movement**: If the instance profile has been customized with more advanced permissions, lateral movement can become even more direct. For example, permissions like ec2-instance-connect:SendSSHPublicKey allow an attacker to push a temporary SSH key to other EC2 instances and gain interactive access, while ssm:SendCommand allows for remote command execution on any instance managed by AWS Systems Manager.[53]

Establishing Persistence

After gaining access and moving laterally, a sophisticated attacker will seek to establish persistence to ensure long-term access to the environment, even if the initial vulnerability is remediated.

- **IAM Backdoors**: The most robust form of persistence is achieved by creating a backdoor in IAM. If the attacker successfully escalates their privileges to an administrative level (as detailed in the next section), they can create a new IAM user with their own set of long-term access keys. Alternatively, they can create a new IAM role with a trust policy that allows it to be assumed by an AWS account under their control.[56]
- **Modifying Launch Configurations (.ebextensions)**: The .ebextensions directory within an application source bundle allows developers to specify custom configurations and run scripts during instance deployment and boot-up.[58] An attacker with write access to the S3 source bucket can add or modify configuration files in this directory to establish persistence on the EC2 instances themselves. Malicious .ebextensions can be crafted to:
  - Create a local user account on the EC2 instance with a known password or SSH key.
  - Install persistent malware, a rootkit, or a reverse shell that initiates a connection to an attacker-controlled server on boot.
  - Modify cron jobs to periodically exfiltrate data or credentials.[59]
- **Application Source Code Backdoor**: The web shell planted during the S3 code tampering attack serves as a simple yet effective persistence mechanism. As long as the malicious code remains in the S3 bucket and is

part of the deployed application version, the attacker retains RCE capabilities.[9]

Advanced Exploitation: Mastering IAM Privilege Escalation

The acquisition of initial IAM credentials from an Elastic Beanstalk instance is a critical milestone, but it is often just the beginning of a deeper compromise. The default roles associated with Elastic Beanstalk, or secondary credentials discovered during enumeration, may have limited permissions. The ultimate goal for an attacker is to escalate these limited privileges to achieve administrative control over the entire AWS account. This is accomplished by identifying and exploiting dangerous combinations of IAM permissions that allow a principal to grant itself more permissions than it currently possesses.[57]

The Principles of IAM Exploitation

The security risk of an IAM permission is rarely atomic; its true danger lies in its interaction with other permissions and the context in which it can be used.[56] An attacker's task is to analyze the policies attached to a compromised user or role and identify these exploitable combinations. While manual analysis is possible, specialized tools can significantly accelerate this process. Pacu's

iam__privesc_scan module, for instance, is designed to automatically test for over 21 known privilege escalation paths, providing a rapid assessment of potential vectors.[32]

The IAM Privilege Escalation Playbook

For a penetration tester, a structured and repeatable methodology is essential for efficiently identifying privilege escalation paths during an engagement. The following table codifies some of the most common and high-impact IAM escalation techniques, synthesizing research from security firms like Rhino Security Labs.[56] It serves as a quick-reference guide to cross-reference a compromised principal's permissions against known attack patterns, transforming theoretical knowledge into an actionable checklist.

Required IAM Permission(s)	Technique Description	Example Exploitation Command (AWS CLI)	Potential Impact & Notes

iam:CreatePolicyVersion	Create a new, more permissive version of an existing policy and set it as default using the --set-as-default flag, bypassing the need for iam:SetDefaultPolicyVersion. [56]	aws iam create-policy-version --policy-arn <arn> --policy-document file://admin.json --set-as-default	Full Administrator Access. The admin.json file would contain a statement like {"Version": "2012-10-17", "Statement":}.
iam:SetDefaultPolicyVersion	Revert a policy to an older, more permissive version that is currently inactive. An attacker can list policy versions to find one with higher privileges. [56]	aws iam set-default-policy-version --policy-arn <arn> --version-id <v1>	Varies; potentially Full Admin Access if a historical version of the policy contained administrative permissions.
iam:PassRole, ec2:RunInstances	Create a new EC2 instance and pass a highly privileged role to it. SSH into the instance and steal the role's credentials from the IMDS. [56]	aws ec2 run-instances --image-id <ami> --iam-instance-profile Name=<privileged-role> --key-name <your-key>	Assumes permissions of the passed role. High potential for admin access if a powerful role (e.g., one with AdministratorAccess) can be passed.
iam:AttachUserPolicy	Attach the AdministratorAccess managed policy directly to the compromised user, immediately granting full administrative rights. [56]	aws iam attach-user-policy --user-name <user> --policy-arn arn:aws:iam::aws:policy/AdministratorAccess	Full Administrator Access. This is one of the most direct escalation paths.
iam:PutUserPolicy	Create a new inline policy for the compromised user that grants full administrative privileges. Inline policies are embedded directly into the user object. [56]	aws iam put-user-policy --user-name <user> --policy-name admin --policy-document file://admin.json	Full Administrator Access. Provides complete control over the permissions granted.

iam:UpdateAssumeRolePolicy	Modify the trust policy of a privileged role to allow the attacker's principal to assume it, thereby gaining access to all permissions attached to that role. [56]	aws iam update-assume-role-policy --role-name <privileged-role> --policy-document file://trust.json	Assumes permissions of the target role. The trust.json file would specify the attacker's user or role ARN as a trusted principal.
lambda:UpdateFunctionCode	Modify the code of an existing Lambda function that has a privileged execution role attached. The new code can be designed to perform malicious actions (e.g., creating an admin user). The attacker then waits for or triggers the function's execution. [56]	aws lambda update-function-code --function-name <func> --zip-file fileb://payload.zip	Assumes permissions of the Lambda's execution role. The impact depends entirely on the permissions of the targeted function's role.
iam:CreateAccessKey	Create new long-term access keys for a different, more privileged user in the account. This is effective if the target user does not already have the maximum of two access keys. [56]	aws iam create-access-key --user-name <privileged-user>	Gains all permissions of the target user. A direct path to impersonation.

A Penetration Tester's Strategic Recommendations

The final and most crucial phase of a penetration test is the delivery of clear, actionable recommendations that enable the client to remediate identified vulnerabilities and improve their overall security posture. For AWS Elastic Beanstalk, these recommendations should address weaknesses across the entire stack, from the application layer to the underlying cloud configuration and IAM policies.

Prioritizing and Reporting Findings

Effective reporting translates technical findings into tangible business risks. Instead of merely listing vulnerabilities, a penetration tester should demonstrate how they can be chained together to create a high-impact attack path.

- **Mapping to Business Impact**: A finding like "SSRF vulnerability identified in the image upload feature" is technical. It should be reframed in terms of business risk: "A vulnerability in the public-facing web application can be exploited to steal cloud infrastructure credentials. These credentials grant an attacker access to the application's source code and the ability to modify sensitive customer data stored in backend databases, posing a critical risk of data breach and service disruption."
- **Chained Exploits**: The report must emphasize the multiplicative effect of chained vulnerabilities. A low-severity SSRF, a medium-severity overly permissive IAM role, and a medium-severity writable S3 bucket, when combined, constitute a critical-severity finding. The report should narrate the entire attack chain, from initial compromise to full account takeover, to illustrate the true risk level.

Key Defensive Countermeasures to Recommend

Based on the common attack vectors detailed in this guide, the following defensive countermeasures should be prioritized in the final report.

- **Enforce IMDSv2**: This is the single most effective control to mitigate SSRF-based credential theft from the EC2 metadata service. The recommendation should be to enforce IMDSv2 across all Elastic Beanstalk environments by configuring the underlying launch templates or configurations to disable IMDSv1. This changes the metadata endpoint to require session-based authentication, breaking most SSRF-to-credential-theft attack chains.[44]
- **Secure Secrets Management**: The practice of storing secrets in environment variables must be strictly prohibited. The report should mandate the adoption of AWS Secrets Manager or AWS Systems Manager Parameter Store for all sensitive data, including database connection strings, API keys, and credentials. The application code should be refactored to retrieve these secrets at runtime using its assigned IAM role, ensuring that secrets are never stored in plaintext within the environment's configuration.[32]
- **Implement Least-Privilege IAM Roles**: The default IAM roles provided by Elastic Beanstalk (aws-elasticbeanstalk-ec2-role) are often overly permissive for a specific application's needs. The recommendation should be to create custom, narrowly-scoped IAM roles for each Elastic Beanstalk environment. These roles should only grant the minimum permissions required for the application to function. For example, if the application only needs to read from a specific S3 bucket, the role should not have s3:PutObject or wildcard S3 permissions. All permissions that could be used for privilege escalation, such as iam:PassRole or iam:AttachUserPolicy, should be explicitly denied unless absolutely necessary.[44]
- **Enable Managed Platform Updates**: To mitigate the risk of exploitation through known vulnerabilities in

outdated platform components, environments should be configured to use "Managed Platform Updates." This feature allows Elastic Beanstalk to automatically apply patches and minor version updates during specified maintenance windows, ensuring the environment does not fall behind on critical security fixes.[44]

- **Harden S3 Bucket Security**: The S3 bucket used by Elastic Beanstalk to store source code and logs is a high-value target. Recommendations should include:
  - Enabling server-side encryption (SSE-S3 or SSE-KMS) to protect the data at rest.[66]
  - Enabling S3 bucket versioning and Multi-Factor Authentication (MFA) Delete to protect against unauthorized modification or deletion of source code bundles.[66]
  - Applying a strict bucket policy that, in addition to the permissions for the instance profile, explicitly denies any unintended public access.
- **Conduct Regular Security Audits and Application Testing**: Organizations should implement a continuous security monitoring and testing program. This includes:
  - Using AWS IAM Access Analyzer to proactively identify and review overly permissive or public-facing IAM roles and policies.[32]
  - Performing regular, automated vulnerability scanning of the web application code itself using Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) tools to identify vulnerabilities like SSRF or RCE before they can be exploited.[69]

Works cited

1. Leaking Secrets via Elastic Beanstalk: CloudGoat Lab | by K0d3-n ..., accessed on September 7, 2025, https://medium.com/@k0d3-n-r011a/leaking-secrets-via-elastic-beanstalk-cloudgoat-lab-c2e59bcce3ef
2. Introduction to AWS Elastic Beanstalk - YouTube, accessed on September 7, 2025, https://www.youtube.com/watch?v=SrwxAScdyT0
3. Amazon Elastic Beanstalk FAQs - Amazon Web Services, accessed on September 7, 2025, https://www.amazonaws.cn/en/elasticbeanstalk/faqs/
4. Understanding concepts in Elastic Beanstalk - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.html
5. Elastic Beanstalk web server environments - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts-webserver.html
6. penetration test - Getting scanned on AWS Elastic BeanStalk ..., accessed on September 7, 2025, https://security.stackexchange.com/questions/199690/getting-scanned-on-aws-elastic-beanstalk
7. What is AWS Elastic Beanstalk? - Hava.io, accessed on September 7, 2025, https://www.hava.io/blog/what-is-aws-elastic-beanstalk
8. Using Elastic Beanstalk with Amazon S3 - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/AWSHowTo.S3.html
9. Exploiting SSRF in AWS Elastic Beanstalk | Claranet Cyber Security, accessed on September 7, 2025, https://www.claranet.com/us/blog/2019-02-01-exploiting-ssrf-aws-elastic-beanstalk
10. Do Elastic Beanstalk Web Server Environment Need A Public Elastic IP | AWS re:Post, accessed on September 7, 2025, https://repost.aws/questions/QUARH0e87FTfaBKPm0BFCmrA/do-elastic-beanstalk-web-server-environment-need-a-public-elastic-ip

11. Managing Elastic Beanstalk instance profiles - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/iam-instanceprofile.html
12. Elastic Beanstalk instance profile - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts-roles-instance.html
13. A repo with steps and examples of how to deploy the Falcon sensor in AWS Elastic Beanstalk - GitHub, accessed on September 7, 2025, https://github.com/CrowdStrike/aws-elastic-beanstalk-integration
14. Python, Amazon's Elastic Beanstalk, AWS & Django App Deployments for Beginners, accessed on September 7, 2025, https://www.youtube.com/watch?v=2N-L7-MAeuc
15. Day 6 of my AWS Pentesting journey — Exploring Elastic Beanstalk | by mauzware | Medium, accessed on September 7, 2025, https://medium.com/@mauzware/day-6-of-my-aws-pentesting-journey-exploring-elastic-beanstalk-adc28ea74f0d
16. Elastic Beanstalk supported platforms - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.platforms.html
17. Container managed policy and EC2 instance role - AWS Elastic Beanstalk, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_docker_ecs_role.html
18. amazon web services - AWS Elastic Beanstalk - Multi Container Docker - Stack Overflow, accessed on September 7, 2025, https://stackoverflow.com/questions/42632362/aws-elastic-beanstalk-multi-container-docker
19. Using the Elastic Beanstalk Go platform - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/go-environment.html
20. Deploying Docker & Go to AWS Beanstalk | by David Dymko - Medium, accessed on September 7, 2025, https://ddymko.medium.com/deploying-docker-go-to-aws-beanstalk-e91207fe21d6
21. Scanning an application in AWS Elastic Beanstalk using Invicti Shark for Java, accessed on September 7, 2025, https://www.invicti.com/support/invicti-shark-scan-aws-java/
22. Using the Elastic Beanstalk Tomcat platform - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/java-tomcat-platform.html
23. AWS Elastic Beanstalk NodeJS and logs - Stack Overflow, accessed on September 7, 2025, https://stackoverflow.com/questions/26972267/aws-elastic-beanstalk-nodejs-and-logs
24. Using the Elastic Beanstalk Node.js platform - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_nodejs.container.html
25. AWS Elastic Beanstalk Environment Variables in Python - Stack Overflow, accessed on September 7, 2025, https://stackoverflow.com/questions/38768549/aws-elastic-beanstalk-environment-variables-in-python
26. Setting up your Python development environment for Elastic Beanstalk, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/python-development-environment.html
27. Using the Elastic Beanstalk Python platform - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-container.html
28. Infrastructure security in Elastic Beanstalk - AWS Documentation, accessed on September 7, 2025,

https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/infrastructure-security.html
29. Enable HTTP header modification for your Application Load Balancer - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticloadbalancing/latest/application/enable-header-modification.html
30. HTTP header modification for your Application Load Balancer - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticloadbalancing/latest/application/header-modification.html
31. HTTP headers and Classic Load Balancers - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/x-forwarded-headers.html
32. Hacking AWS Beanstalk — Cloudgoat Scenario | by Aravind S V, accessed on September 7, 2025, https://aws.plainenglish.io/hacking-aws-beanstalk-cloudgoat-scenario-4da1fac6a4cf
33. New Pacu Module: Secret Enumeration in Elastic Beanstalk - Rhino Security Labs, accessed on September 7, 2025, https://rhinosecuritylabs.com/tools/new-pacu-module-enumerating-elastic-beanstalk/
34. list_platform_versions — Boto3 Docs 1.26.85 documentation - AWS, accessed on September 7, 2025, https://boto3.amazonaws.com/v1/documentation/api/1.26.85/reference/services/elasticbeanstalk/client/list_platform_versions.html
35. list-platform-versions — AWS CLI 2.8.7 Command Reference, accessed on September 7, 2025, https://awscli.amazonaws.com/v2/documentation/api/2.8.7/reference/elasticbeanstalk/list-platform-versions.html
36. ListPlatformVersions - AWS Elastic Beanstalk, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/api/API_ListPlatformVersions.html
37. aws elasticbeanstalk describe-platform-version - Fig.io, accessed on September 7, 2025, https://www.fig.io/manual/aws/elasticbeanstalk/describe-platform-version
38. Hacking AWS Elastic Beanstalk: CloudGoat's beanstalk_secrets Lab Walkthrough - Medium, accessed on September 7, 2025, https://medium.com/@vulnxpert/hacking-aws-elastic-beanstalk-cloudgoats-beanstalk-secrets-lab-walkthrough-a21135c6579e
39. SSRF to AWS Metadata Exposure: How Attackers Steal ... - Resecurity, accessed on September 7, 2025, https://www.resecurity.com/blog/article/ssrf-to-aws-metadata-exposure-how-attackers-steal-cloud-credentials
40. AWS IMDSv1 Vulnerability Exposed: Insights from TotalCloud - Qualys Blog, accessed on September 7, 2025, https://blog.qualys.com/vulnerabilities-threat-research/2024/09/12/totalcloud-insights-unmasking-aws-instance-metadata-service-v1-imdsv1-the-hidden-flaw-in-aws-security
41. Misconfiguration Spotlight: Securing the EC2 Instance Metadata Service, accessed on September 7, 2025, https://securitylabs.datadoghq.com/articles/misconfiguration-spotlight-imds/
42. Server Side Request Forgery (SSRF) and AWS EC2 instances after Instance Meta Data Service version 2(IMDSv2) | by Riyaz Walikar | Appsecco, accessed on September 7, 2025, https://blog.appsecco.com/server-side-request-forgery-ssrf-and-aws-ec2-instances-after-instance-meta-data-service-version-38fc1ba1a28a
43. IMDSv1 Credential Access - AWS Builder Center, accessed on September 7, 2025,

https://builder.aws.com/content/2q1QWCuHPXjCkkjA1cPBHjq7PT7/imdsv1-credential-access
44. Security best practices for Elastic Beanstalk - AWS Documentation, accessed on September 7, 2025, https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/security-best-practices.html
45. Upgrade platform versions in Elastic Beanstalk | AWS re:Post, accessed on September 7, 2025, https://repost.aws/knowledge-center/elastic-beanstalk-upgrade-platform
46. Elastic Beanstalk PHP (AL2) includes old Ruby packages : r/aws - Reddit, accessed on September 7, 2025, https://www.reddit.com/r/aws/comments/18ypq8q/elastic_beanstalk_php_al2_includes_old_ruby/
47. Amazon Linux Security Center - CVE List, accessed on September 7, 2025, https://explore.alas.aws.amazon.com/
48. Remote Code Execution (RCE) | Types, Examples & Mitigation | Imperva, accessed on September 7, 2025, https://www.imperva.com/learn/application-security/remote-code-execution/
49. What is remote code execution? - Cloudflare, accessed on September 7, 2025, https://www.cloudflare.com/learning/security/what-is-remote-code-execution/
50. Remote Code Execution (RCE) Explained in Detail - Splunk, accessed on September 7, 2025, https://www.splunk.com/en_us/blog/learn/rce-remote-code-execution.html
51. Potential Remote Code Execution via Web Server | Prebuilt detection rules reference, accessed on September 7, 2025, https://www.elastic.co/docs/reference/security/prebuilt-rules/rules/linux/persistence_linux_shell_activity_via_web_server
52. AWSElasticBeanstalkWebTier - AWS Managed Policy, accessed on September 7, 2025, https://docs.aws.amazon.com/aws-managed-policy/latest/reference/AWSElasticBeanstalkWebTier.html
53. EC2-Instance-Connect Lateral Movement Strategy for Data Exfiltration - Uptycs, accessed on September 7, 2025, https://www.uptycs.com/blog/ec2-instance-connect-lateral-movement-strategy-and-tactics-for-data-exfiltration
54. Lateral movement risks in the cloud and how to prevent them – Part 1: the network layer (VPC) | Wiz Blog, accessed on September 7, 2025, https://www.wiz.io/blog/lateral-movement-risks-in-the-cloud-and-how-to-prevent-them-part-1-the-network-layer
55. AWSElasticBeanstalkWorkerTier - AWS Managed Policy, accessed on September 7, 2025, https://docs.aws.amazon.com/aws-managed-policy/latest/reference/AWSElasticBeanstalkWorkerTier.html
56. AWS IAM Privilege Escalation – Methods and Mitigation - Rhino Security Labs, accessed on September 7, 2025, https://rhinosecuritylabs.com/aws/aws-privilege-escalation-methods-mitigation/
57. AWS IAM Exploitation - Security Risk Advisors, accessed on September 7, 2025, https://sra.io/blog/aws-iam-exploitation/
58. How to use persistent storage with AWS Elastic Beanstalk Windows deployment., accessed on September 7, 2025, https://aws.amazon.com/blogs/modernizing-with-aws/how-to-use-persistent-storage-with-aws-elastic-beanstalk-windows-deployment/
59. How to set EBS root volume to persist for an EC2 instance within Elastic Beanstalk using Terraform - Stack Overflow, accessed on September 7, 2025,

https://stackoverflow.com/questions/63018583/how-to-set-ebs-root-volume-to-persist-for-an-ec2-instance-within-elastic-beansta

60. AWS IAM Privilege Escalation - Methods and Mitigation - Part 2 - Rhino Security Labs, accessed on September 7, 2025, https://rhinosecuritylabs.com/aws/aws-privilege-escalation-methods-mitigation-part-2/
61. Privilege escalation with IAM on AWS | SideChannel – Tempest, accessed on September 7, 2025, https://www.sidechannel.blog/en/privilege-escalation-with-iam-on-aws/
62. Investigating Privilege Escalation Methods in AWS - Bishop Fox, accessed on September 7, 2025, https://bishopfox.com/blog/privilege-escalation-in-aws
63. Exploiting IAM security misconfigurations - Sysdig, accessed on September 7, 2025, https://www.sysdig.com/blog/iam-security-misconfiguration
64. Configure secrets as Elastic Beanstalk environment variables | AWS re:Post, accessed on September 7, 2025, https://repost.aws/questions/QU31LdCFwTQA2-BYvqw4PW5g/configure-secrets-as-elastic-beanstalk-environment-variables
65. AWS Elastic Beanstalk environment managed platform updates are not enabled, accessed on September 7, 2025, https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-general-policies/bc-aws-340
66. Top 11 AWS Misconfigurations and How to Avoid Them - CrowdStrike.com, accessed on September 7, 2025, https://www.crowdstrike.com/en-us/cybersecurity-101/cloud-security/aws-misconfigurations/
67. Hardening the security of your AWS Elastic Beanstalk Application the Well-Architected way, accessed on September 7, 2025, https://aws.amazon.com/blogs/security/hardening-the-security-of-your-aws-elastic-beanstalk-application-the-well-architected-way/
68. TotalCloud Insights: Hidden Risks of Amazon S3 Misconfigurations - Qualys Blog, accessed on September 7, 2025, https://blog.qualys.com/vulnerabilities-threat-research/2023/12/18/hidden-risks-of-amazon-s3-misconfigurations
69. 7 Simple AWS Security Best Practices to Follow | BeyondTrust, accessed on September 7, 2025, https://www.beyondtrust.com/blog/entry/aws-security-best-practices
70. A Complete Guide on AWS Penetration Testing - Astra Security, accessed on September 7, 2025, https://www.getastra.com/blog/security-audit/aws-penetration-testing/