BlueRock AWS CloudFormation Deployment - v1.2.2

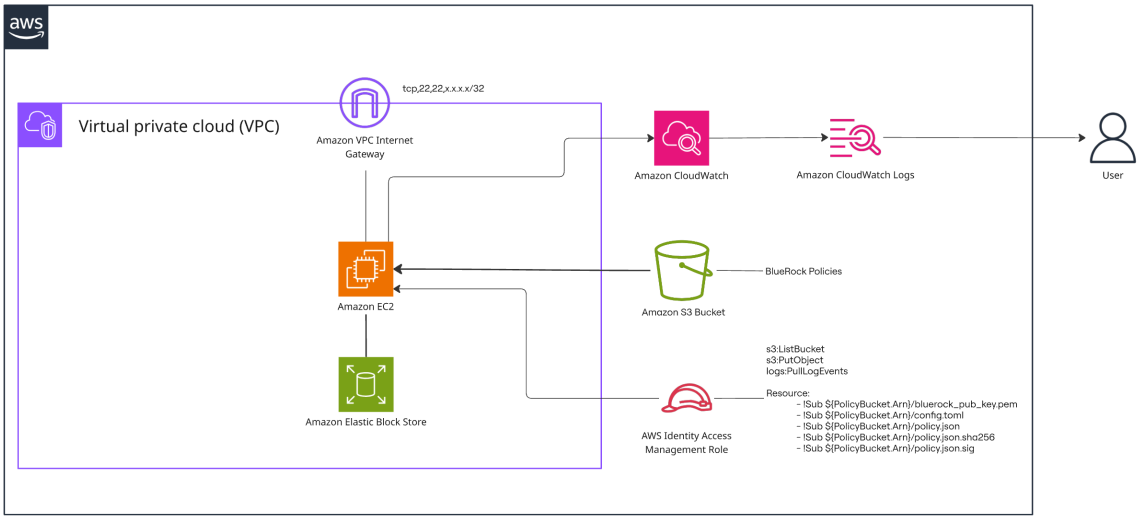
BlueRock AWS CloudFormation Deployment

Introduction

The following is a guide that shows how to create and deploy a BlueRock Node using a CloudFormation template in AWS.

Architecture Diagram

BlueRock EC2 Deployment on AWS



Above is the architecture created by the CloudFormation Template. It contains the following components:

- The BlueRock Node Instance. This instance and its workloads are protected by BlueRock, it contains the following:
  - BlueRock Rule Processing Engine: This container manages the node's policy and collects information about the node for rule enforcement. BlueRock policies can be configured for application and container runtime as well as process and file level controls.
  - Trex: Trex is an internal tool that turns simple json policy files into signed BlueRock consumable policy files. After writing a new policy file it needs to be processed by Trex before being uploaded to the policy bucket
  - OTel Collector: BlueRock manages its logs using Open Telemetry Receivers, Processors and Collectors. An intermediate collector has been placed as a container on this instance to allow for ease of access in the log management for this marketplace listing. All BlueRock Logs are sent through this intermediary collector on their way to Cloudwatch.
- Additional AWS services: This template utilizes additional Amazon services for configuration and event monitoring
  - Amazon S3: This service is used to store signed policies in a BlueRock Policy S3 Bucket
  - Amazon CloudWatch: BlueRock sends events to a CloudWatch Log Group via the OTel Collector

Installation

Prerequisites

The following NodeInstanceTypes are supported with BlueRock. Larger instance sizes should also be compatible, but the below list has been validated:

```
t3.2xlarge
t3.xlarge
t3.large
t3.medium
t3.micro
t3.nano
t3.small
m5.2xlarge
m5.xlarge
m5.large
r5.2xlarge
r5.xlarge
r5.large
r5n.2xlarge
r5n.xlarge
r5n.large
i4i.2xlarge
i4i.xlarge
i4i.large
d3.2xlarge
d3.xlarge
```

The BlueRock Installation requires customers launch the BlueRock EC2 deployment from AWS Marketplace. Free private offers can be made available upon request.

- The BlueRock Amazon Linux 2023 EC2 listing can be found here.

IAM Role and Policy Documentation

This CloudFormation template creates an IAM role (`InstanceRole`) and an associated IAM policy (`InstancePolicy`). The purpose of these resources is to grant specific permissions to an Amazon EC2 instance, allowing it to interact with other AWS services in a secure and controlled manner.

`InstanceRole`

The `InstanceRole` is an IAM role designed to be assumed by an Amazon EC2 instance.

- **Purpose**: This role grants the EC2 instance the necessary permissions to perform its designated tasks by allowing it to assume this role.
- **Trusted Entity**: The role's trust policy specifies that only the EC2 service (`ec2.amazonaws.com`) can assume this role. This is a standard practice for creating IAM roles for EC2 instances.
- **Managed Policies**: This role includes two AWS managed policies:
  - `arn:aws:iam::aws:policy/AmazonEC2ReadOnlyAccess`: This policy grants read-only access to Amazon EC2 resources. This allows the instance to describe EC2 resources, which can be useful for inventory or monitoring purposes without allowing any modifications.
  - `arn:aws:iam::aws:policy/AWSMarketplaceManageSubscriptions`: This policy provides permissions to manage AWS Marketplace subscriptions. This allows the instance to subscribe to and unsubscribe from AWS Marketplace products.

`InstancePolicy`

The `InstancePolicy` is a custom IAM policy that grants additional, more specific permissions to the `InstanceRole`.

- **Purpose**: This policy defines fine-grained permissions required by the application running on the EC2 instance to interact with Amazon S3 and Amazon CloudWatch Logs.
- **Permissions**: The policy contains the following statements:
  - **Statement 1: General Permissions**
    - **`s3:ListBucket`**: This permission allows the instance to list the objects within an S3 bucket. This is often required to iterate through the contents of a bucket.
    - **`s3:PutObject`**: This permission allows the instance to upload new objects to an S3 bucket.
    - **`logs:PutLogEvents`**: This permission allows the instance to upload log events to a CloudWatch Logs stream. This is essential for applications that need to centralize their logging.
    - **Resource**: The resource is set to `*`, which means these actions are allowed on all resources of the respective service. While this provides broad access, the subsequent statement narrows down the `s3:PutObject` permission.
  - **Statement 2: Specific S3 Upload Permissions**
    - **`s3:PutObject`**: This part of the policy further refines the `PutObject` permission. It restricts the upload of specific files to a designated S3 bucket (`${PolicyBucket.Arn}`). The files that can be uploaded are:
      - `bluerock_pub_key.pem`
      - `config.toml`
      - `policy.json`
      - `policy.json.sha256`
      - `policy.json.sig`
    - **Resource**: The permissions are scoped to the specific S3 bucket and the exact file names, following the principle of least privilege by ensuring the instance can only write these specific files to the intended location.

To use this Cloud Formation template, ensure that the calling entity has the permissions necessary to call for `CAPABILITY_NAMED_IAM` capabilities. Additionally, ensure that an AWS managed ssh key is present in the region. As a parameter of the template, a key managed by the EC2 service is needed for developers to access the nodes. Developers should have access to the name of their key and the private and public key files.

Using CloudFormation

Through the AWS Web Console

1. Navigate to the CloudFormation Service Page
2. Select **Create stack** → **With new resources (standard)**
3. Select **Choose an existing template**
    a. Specify template source by selecting the **Upload a template file**
4. Upload the BlueRock CloudFormation Template
5. Select **Next**
6. Fill in all template parameters (see Parameter table below)
7. Select **Next**
8. Under **Capabilities** Acknowledge the creation of IAM Roles / Policies (see Policy Table Below)
9. Select **Next**
10. Confirm Stack creation and **Submit**

Through the AWS CLI

Make sure the CLI is installed and has privileges to specify `CAPABILITY_NAMED_IAM`

Call `create-stack` specifying the parameter file, template file and capabilities

```
aws cloudformation create-stack --stack-name <string-name> --template-body file://<template-file-location> --capabilities CAPABILITY_NAMED_IAM --parameters file://<json-param-file-location>
```

Example Parameters File

```
[
  {
    "ParameterKey": "AllowIp",
    "ParameterValue": "<your_ip_address>/32" or "0.0.0.0/0"
  },
  {
    "ParameterKey": "ImageId",
    "ParameterValue": <this value is predefined by region by AWS>
  },
  {
    "ParameterKey": "NodeInstanceType",
    "ParameterValue": "t3.medium"
  },
  {
    "ParameterKey": "Prefix",
    "ParameterValue": "<user-defined prefix string for provisioned
resources>"
  },
  {
    "ParameterKey": "SampleHostName",
    "ParameterValue": "<user-defined hostname identifer>"
  },
  {
    "ParameterKey": "SshKeyName",
    "ParameterValue": "<key pair for EC2 access>"
  },
]
```

CloudFormation Template Parameters

Parameter Name	Description
`AllowIp`	Developer's IP address to whitelist in the BlueRock Instance for SSH access or 0.0.0.0/0
`NodeInstanceType`	Size of BlueRock node to be created.
`NodeAmi`	Specified by AWS per region
`Prefix`	Unique Identifier appended to AWS resource names
`SampleHostName`	Unique Identifier for BlueRock host name
`SshKeyName`	Name of AWS SSH key pair, used for authorizing access to the BlueRock Instance

This stack takes about 2-5 minutes to build.

Installation Validation Checks

SSH into BlueRock Instance

Once the stack has completed, select it (**CloudFormation → <Stack-Name>**) and view the **Resources** tab. Here you can search for the `NodeInstance` instance. Connect to `NodeInstance` via an SSH command using the key specified in `SshKeyName`.

SSH Command

```
ssh -i <priv-key> ec2-user@<instance-pub-ip>
```

Check Services

The BlueRock Services are initialized through the `uc-docker.service` service. Ensure that the service and the Rule Engine container are running and enabled. Additionally, use the provided script to view the logs exported by the Rule Engine.

```
$sudo systemctl status uc-docker.service
$ docker ps
CONTAINER ID   IMAGE
COMMAND                  CREATED       STATUS       PORTS     NAMES
5b5e00d1881f   ultracontrol:latest
"/opt/bluerock/sbin/…"   7 hours ago   Up 7 hours             uc
e0186c875227   public.ecr.aws/aws-observability/aws-otel-
collector:latest   "/awscollector --con…"   7 hours ago   Up 7 hours
otel-collector

$/opt/bluerock/bin/uc-docker.sh logs
```

Finally, ensure the OTEL collector service is running and enabled.

```
$docker logs otel-collector
```

Policy Configuration

To enable the enforcement of policies, BlueRock Instances need a reachable policy file in place. In this architecture, we will load the policy file into the CF generated S3 bucket.

First we will create the signed policy file on the BlueRock Instance.

Once connected, navigate to `~/policy` where a sample policy (`bru_policy.json`) with all protection mechanisms set to observe mode is present. Once the policy is modified we need to sign it using Trex.

```
$ python3 /opt/bluerock/trex/trex.py <policy-file>
$ tar xvf <policy-file>.tar
```

- Ensure that the `bru-venv` is activated when using any python commands

  ```
  $ source /home/ec2-user/build/bru-venv/bin/activate
  ```

This will generate a tar file containing the new policy, the policy's signature and a sha256 sum of the policy. All three of these files need to be uploaded to the BlueRock Policy Bucket.

The Services Instance also generates a new public key on creation, this key is specified in the `trex.toml` file located in the `policy` directory. This public key also must be uploaded to the BlueRock Policy Bucket.

Pushed Item	Description
policy.json	Json version of buv_policy.yaml, this includes additional configs supplied by Trex
policy.json.sig	signature file for policy.json, needed for verifying authenticity of policy file
policy.json.sha256	sha256 sum of policy.json, needed for verifying integrity of policy file

```
aws s3 sync . s3://<bluerock-cfg-s3-bucket-name> --exclude "*" --
include "policy.json*"
aws s3 cp ./bluerock_pub_key.pem s3://<bluerock-cfg-s3-bucket-name>
```

These policy files take ~10 mins to propagate to each node.

Refer to the Configuring BlueRock Security Policies section for instructions on editing and tuning policies.

Any time the policy file is updated the tar needs to be re-created and pushed to the policy bucket.

Configuring BlueRock Security Policies

This guide will walk you through understanding and configuring security policies for BlueRock. Policies allow you to fine-tune how the system detects and blocks various security events and behaviors.

Introduction to Policies

BlueRock policies are a set of rules defining how different security mechanisms operate on your protected systems. Each mechanism targets different events or attack methods, such as container drift, process execution, file access, or language-specific protections (like Python or Java) to support both detection and blocking of malicious actions.

Policy rules can be configured in either a detect only mode or a detect and block mode. By customizing these policies, you can:

- Enable or disable specific security protections.
- Define whether threats are simply detected or actively remediated (blocked).
- Define the blocking mechanism - inline synchronous or asynchronous.
- Specify exceptions and allow-lists for legitimate activities.
- Tailor the sensitivity and behavior of security controls to match your environment's needs.

Policy Structure

The security policy is composed in a JSON file that has distinct rule sections or “stanzas.” Each stanza configures a particular security feature. A default JSON policy file is located in the Appendix section of this document.

For example, you'll find stanzas like:

- container_drift: Monitors and controls changes within running containers.
- process_exec: Controls which binaries are allowed to execute.
- sensitive_file_access: Detects or blocks unauthorized access to critical files.
- python_sensor: Provides specific protections for Python applications.
- java_sensor: Provides specific protections for Java applications.

Special Stanzas:

- common and kernel_common: These stanzas are not security mechanisms themselves. Instead, they are used to define common configuration data (like lists of known interpreters, web servers, or shells) that can be referenced and used by multiple other security policy.

Below is an example of a rule for NSenter for namespace changes. This policy rule controls whether non-system daemons can switch PID namespaces (a method for breaking out of containers):

```
"nsenter": {
    "enable": true,
    "remediate": true,
    "inline": true
  },
```

Common Configuration Options in Policy Rules

Most rules share a common set of configuration options that control their basic behavior: enable, remediate and inline. Below are explanations of these rule options.

1. **enable**
   - **Purpose:** Enables or disables the specific security detection mechanism.
   - **Values:** `true` (enabled) or `false` (disabled).
   - **Default:** Typically false for most mechanisms, meaning they are off by default and you must explicitly enable them.
2. **remediate**
   - **Purpose:** Determines if the mechanism should take action to block a detected event, or if it should only be logged.
   - **Values:** true (block asynchronously) or false (detect only). The malicious action might briefly occur, but it is then terminated or remediated shortly afterward.
   - **Note:** This option is only considered if enable is set to true.
3. **inline**
   - **Purpose:** Specifies if inline synchronous blocking should be applied to a detected event.
   - **Values:** true (enable inline blocking). The action is blocked *before* it can complete. This provides the strongest protection mechanism. false: (Asynchronous Remediation). The malicious action might briefly occur, but it is then terminated or remediated shortly afterward.
   - **Note:** Inline blocking is only enabled if both inline and remediate are set to true. Some mechanisms may only operate in one mode (e.g., always inline or always asynchronous), and this option might not be available for specific rules.

Below is an example of a Java deserialization rule set to block inline.

```
"java_deserialization": {
      "enable": true,
      "remediate": false,
      "inline": true,
      "deny_list": [
        ""
      ],
      "allow_list": [
        ""
      ]
    },
```

Rule Type Descriptions

Application Runtime - Language-Specific Sensor Rules

Python Sensor: The Python Sensor supports detection and blocking capabilities for a range of attack methods that include the following:

- *Python Pickle Deserialization* - Detects and/or blocks data deserialization which can be used to pass commands and execute exploits on a given system. The **python_sensor** and **pickle** rules configure this mechanism.
- *Python Code Execution Control* - Detects and/or blocks new processes being created and executed. The **python_execs** rule configures this mechanism.
- *Path Traversal* - Detects and/or blocks attempts to navigate to alternate directories on host using path access methods that may circumvent file permissions. The **pathtraversal** rule configures this mechanism.
- *Import and Load Detection* - Detects and/or blocks the import of files, modules, packages, and classes with path, version and hash information. The **python_imports** rule is used to configure this mechanism. **python_loads** is used to configure identification of new code imported into an existing process.

Java Sensor: The Java Sensor supports detection and blocking capabilities for a range of attack methods that include the following:

- *Java Deserialization* - Detects and/or blocks data deserialization which can be used to pass commands and execute exploits on a given system. The **java_deserialization** rule configures this mechanism.
- *Java Code Execution Control* - Detects and/or blocks code execution associated with deserialization or network activity. The **java_exec** rule configures this mechanism.
- *Network Activity Detection and Control* - Detects and/or blocks threads where network activity is associated with an active thread through a class invocation. Uses the **java_reflection** rule to configure this mode. **java_exec** also identifies traces of network activity.
- *File Open Control* - Detects and/or blocks new or anomalous file opens that may be associated with path traversal and/or network activity. The **java_file_open** rule is used to configure

Example:

```
"python_sensor": {
    "enable": true,
    "pickle": {
      "enable": true,
      "remediate": false,
      "inline": true,
      "functions_deny_list": [
        "builtins.exec",
        "posix.system",
        "socket.socket"
      ],
      "functions_allow_list": null
    },
    "pathtraversal": {
      "enable": true,
      "remediate": false,
      "inline": true,
      "deny_list": [
        "/etc/passwd",
        "/etc/shadow"
      ]
    },
    "imports": {
      "enable": true,
      "remediate": false,
      "inline": true,
      "deny_list": [
        ""
      ]
    },
    "execs": {
      "enable": true,
      "remediate": false,
      "deny_list": [
        ""
      ],
      "inline": true,
      "allow_list": [
        ""
      ]
    },
    "loads": {
      "enable": true,
      "remediate": false,
      "inline": true,
      "deny_list_dlsym": [
        ""
      ]
```

```
      ],
      "deny_list_dlopen": [
        ""
      ],
      "deny_list_load": [
        ""
      ]
    },
    "urllib": {
      "enable": true,
      "remediate": false,
      "inline": true
    },
    "url": {
      "enable": true,
      "remediate": false,
      "inline": true
    },
    "misc": {
      "enable": true,
      "remediate": false,
      "inline": true
    }
  },
  "java_sensor": {
    "enable": true,
    "java_class_load": {
      "enable": true,
      "remediate": false,
      "inline": true
    },
    "java_file_open": {
      "enable": true,
      "remediate": false,
      "inline": true,
      "deny_list": [
        "/etc/passwd",
        "/etc/shadow"
      ]
    },
    "java_deserialization": {
      "enable": true,
      "remediate": false,
      "inline": true,
      "deny_list": [
        ""
      ],
      "allow_list": [
        ""
```

```
      ]
    },
    "java_exec": {
      "enable": true,
      "remediate": false,
      "inline": true
    },
    "java_reflection": {
      "enable": true,
      "remediate": false,
      "inline": true
    },
    "java_misc": {
      "enable": true,
      "remediate": false
    }
  }
```

Container Runtime Detection and Blocking Rules

- **Process IO Bind Socket Control (Remote Shell Control)**: Provides a detection and blocking mechanism to identify processes binding sockets to their standard input/output. This method is often used by attackers to establish reverse shell command and control over a host.

  Settings:

  - **enable**: Turn detection on or off.
  - **remediate**: Turn asynchronous blocking on or off.
  - **inline**: Turn synchronous blocking on or off (**remediate** must be set to **true** for inline blocking to take effect).
  - **transitive**: Optional method to detect transitive piping to launch other shells.
  - **explicit_deny**: List specific shells or interpreters whose I/O binding you want to block.

Example:

```
"process_io_bind_sock_control": {
    "enable": true,
    "remediate": true,
    "inline": true,
    "transitive": false,
    "explicit_deny": [
        "/sh",
        "/dash",
        "/bash",
        "/ksh",
        "/ash",
        "/zsh",
        "/python",
"/perl"
],
```

- **Forced Memory Access**: This policy rule provides protection against forced memory access. This mechanism detects and/or prevents processes from interacting with other processes and blocks read/write access to other processes memory address spaces. NOTE: This will prevent debuggers from being able to attach to processes.
  - **enable**: Turn detection on or off.
  - **remediate**: Turn asynchronous blocking on or off.
  - **inline**: Turn synchronous blocking on or off (**remediate** must be set to **true** for inline blocking to take effect).
  - **prevent_writes_only**: By default, with the **true** setting only memory *writes* are blocked. Set to **false** to block all memory access.

Example:

```
"forced_mem_access": {
    "enable": false,
    "remediate": false,
    "inline": true,
    "prevent_writes_only": true
  },
```

Container Drift and Capabilities Policy Rules

- **Container Drift**: This rule detects and/or blocks when a binary or script is executed within a container that did not exist in the container at container start time.
  - **enable**: Turn detection on or off.
  - **remediate**: Turn asynchronous blocking on or off.
  - **inline**: Turn synchronous blocking on or off (remediate must be set to true for inline blocking to take effect).
  - **container_exception_list**: List of containers that are allowed to drift.
  - **pipe_process_exception_list**: List of processes that can pipe info to an interpreter.

Example:

```
"container_drift": {
    "enable": false,
    "remediate": false,
    "inline": true,
    "container_exception_list": [],
    "pipe_process_exception_list": [
        "docker-entrypoi",
        "containerd-shim",
        "bpftrace"
    ]
},
```

- **Container Capabilities**: This rule allows you to configure which capabilities a container object may leverage. Typical scenarios may be to enforce least privilege and privilege escalations. The blocking mode for this rule is always asynchronous.
  - **enable**: Turn detection on or off.
  - **remediate**: Turn asynchronous blocking on or off.
  - **caps_denied**: List capabilities that are forbidden for *all* containers.
  - **explicit_allow**: Allow specific containers to use capabilities, even if they're generally denied.

Example:

```
"container_capabilities": {
    "enable": false,
    "remediate": false,
    "caps_denied": [CAP_SYS_ADMIN, CAP_KILL],
    "explicit_allow": []
},
```

Supported Container Capabilities

System Administration

`CAP_LINUX_IMMUTABLE`

Allows setting and clearing the immutable and append-only attributes on files. These attributes provide additional protection against file modification or deletion.

`CAP_SYS_MODULE`

Allows loading and unloading kernel modules. This is a powerful capability that enables dynamic modification of kernel functionality and should be granted with extreme caution.

`CAP_SYS_RAWIO`

Allows performing raw I/O operations including accessing device memory, modifying NUMA memory policies, and accessing PCI configuration space.

`CAP_SYS_CHROOT`

Allows using the chroot() system call to change the root directory. This capability is essential for creating isolated filesystem environments and containers.

`CAP_SYS_PTRACE`

Allows tracing arbitrary processes using ptrace(). This capability enables debugging tools, process monitoring, and runtime analysis of other processes.

`CAP_SYS_PACCT`

Allows configuring process accounting, which tracks resource usage and process execution statistics for system monitoring and billing purposes.

`CAP_SYS_ADMIN`

Provides a broad range of system administration operations including mounting filesystems, configuring swap, setting hostname, and many other privileged operations. Often called the "new root" capability.

`CAP_SYS_BOOT`

Allows rebooting the system and loading new kernels for later execution. This capability provides control over system shutdown and restart operations.

`CAP_SYS_NICE`

Allows modifying process scheduling priorities and policies. This capability enables changing nice values, real-time scheduling parameters, and CPU affinity settings.

`CAP_SYS_RESOURCE`

Allows exceeding various system resource limits such as file size limits, process limits, and memory limits. Provides control over resource quota enforcement.

`CAP_SYS_TIME`

Allows setting the system clock and real-time hardware clock. This capability is essential for time synchronization services and system time management.

`CAP_SYS_TTY_CONFIG`

Allows configuring terminal devices including setting terminal attributes, configuring virtual consoles, and managing TTY-related operations.

File and Directory Permissions

`CAP_CHOWN`

Allows changing file ownership (user and group) of files and directories. This capability enables the chown() and fchown() system calls regardless of the current user's ownership of the file.

`CAP_DAC_OVERRIDE`

Bypasses file read, write, and execute permission checks. This capability allows accessing files regardless of their permission bits, effectively overriding discretionary access control (DAC) restrictions.

`CAP_DAC_READ_SEARCH`

Bypasses file read permission checks and directory read and execute permission checks. More limited than CAP_DAC_OVERRIDE as it only affects read operations and directory traversal.

`CAP_FOWNER`

Bypasses permission checks on operations that normally require the file owner's permissions, such as changing file permissions, setting extended attributes, or modifying file timestamps.

`CAP_FSETID`

Allows setting the setuid and setgid bits on files, and prevents the kernel from clearing these bits when a file is modified. Essential for creating executable files that run with elevated privileges.

Process and User Management

`CAP_KILL`

Allows sending signals to processes owned by other users. Without this capability, processes can only send signals to processes with the same effective user ID.

`CAP_SETGID`

Allows changing the group ID of the calling process and setting supplementary group IDs. This capability is essential for processes that need to assume different group identities.

`CAP_SETUID`

Allows changing the user ID of the calling process. This capability enables processes to switch between different user identities, commonly used by authentication services and privilege-dropping applications.

`CAP_SETPCAP`

Allows modifying process capabilities. This capability enables a process to grant or remove capabilities from other processes, providing fine-grained privilege management.

Network Operations

`CAP_NET_BIND_SERVICE`

Allows binding to privileged ports (ports numbered less than 1024). Traditionally, only root could bind to these well-known service ports like HTTP (80) and HTTPS (443).

`CAP_NET_BROADCAST`

Allows making socket broadcasts and listening to multicast traffic. This capability is required for network discovery protocols and broadcast-based communication.

`CAP_NET_ADMIN`

Provides extensive network administration privileges including configuring network interfaces, managing routing tables, setting firewall rules, and modifying network namespaces.

`CAP_NET_RAW`

Allows creating raw sockets and packet sockets. This capability is required for network diagnostic tools, custom protocol implementations, and low-level network programming.

Inter-Process Communication

`CAP_IPC_LOCK`

Allows locking memory pages into RAM (preventing them from being swapped to disk) and exceeding resource limits on memory locking operations.

`CAP_IPC_OWNER`

Bypasses permission checks for System V IPC operations including shared memory, semaphores, and message queues. Allows modifying IPC objects owned by other users.

File System Operations

`CAP_MKNOD`

Allows creating special files including device files, named pipes (FIFOs), and sockets using the mknod() system call.

`CAP_LEASE`

Allows establishing file leases, which provide notifications when other processes attempt to open or truncate files. Used for implementing file locking and caching mechanisms.

Security and Auditing

`CAP_AUDIT_WRITE`

Allows writing records to the kernel audit log. This capability enables applications to generate audit events for security monitoring and compliance purposes.

`CAP_AUDIT_CONTROL`

Allows configuring audit subsystem behavior including enabling/disabling auditing, changing audit rules, and modifying audit configuration parameters.

`CAP_AUDIT_READ`

Allows reading from the audit log via multicast netlink socket. This capability enables audit log analysis tools and security monitoring applications.

`CAP_SETFCAP`

Allows setting file capabilities on executable files. This capability enables the creation of capability-aware binaries that can run with specific privileges.

Mandatory Access Control

`CAP_MAC_OVERRIDE`

Allows overriding Mandatory Access Control (MAC) restrictions. This capability bypasses security policies enforced by systems like SELinux or AppArmor.

`CAP_MAC_ADMIN`

Allows configuring or changing Mandatory Access Control policies. This capability enables modification of MAC security rules and policy management.

System Logging and Power Management

`CAP_SYSLOG`

Allows performing privileged syslog operations including reading from kernel message ring buffer and controlling console log level.

`CAP_WAKE_ALARM`

Allows setting wake-up alarms that can wake the system from suspend or hibernation states. Used by power management and scheduling applications.

`CAP_BLOCK_SUSPEND`

Allows blocking system suspend and hibernation. This capability enables applications to prevent the system from entering sleep states when critical operations are running.

Advanced Capabilities

`CAP_PERFMON`

Allows using performance monitoring and observability tools including perf events, BPF programs for performance analysis, and accessing performance counters.

`CAP_BPF`

Allows loading BPF (Berkeley Packet Filter) programs and creating BPF maps. This capability enables advanced networking, tracing, and security applications.

`CAP_CHECKPOINT_RESTORE`

Allows using checkpoint/restore functionality to save and restore process state. This capability enables container migration and process state management features.

Security Note: These capabilities should be granted with careful consideration of the principle of least privilege. Each capability represents significant system access that could be misused if granted unnecessarily.

- **NSEnter**: Controls whether non-system daemons can switch PID namespaces (a method for breaking out of containers).
  - **enable**: Turn detection on or off.
  - **remediate**: Turn asynchronous blocking on or off.
  - **inline**: Turn synchronous blocking on or off (remediate must be set to true for inline blocking to take effect).
  - **allow_list_comm**: List processes that *can* switch namespaces.
- **Container Socket Protect**: Protects important files like `docker.sock` from being accessed by container processes.
  - **enable**: Turn detection on or off.
  - **remediate**: Turn asynchronous blocking on or off.
  - **inline**: Turn synchronous blocking on or off (remediate must be set to true for inline blocking to take effect)
  - **protected_file_paths**: List the socket paths you want to protect.
- **Rogue Container**: Identifies containers that started without your environment knowing about them (like Kubernetes). Blocking is always asynchronous.
  - **enable**: Turn detection on or off.
  - **remediate**: Turn asynchronous blocking on or off.
  - **allow_list_comm**: List legitimate container programs that are allowed to start.

Process and File Security Rules

- **Process Exec**: This rule specifies where programs are allowed to execute from.
  - **enable**: Turn detection on or off.
  - **remediate**: Turn asynchronous blocking on or off.
  - **inline**: Turn synchronous blocking on or off (remediate must be set to true for inline blocking to take effect)
  - **allowed_paths**: List directories where executables *can* run.
  - **excluded_paths**: List specific directories *within* allowed paths that are actually forbidden.

Example:

```
"process_exec": {
    "enable": false,
    "remediate": false,
    "inline": true,
    "allowed_paths": [
      "/bin/",
      "/sbin/",
      "/usr/bin/",
      "/usr/sbin/",
      "/usr/local/bin/",
      "/usr/local/sbin/",
      "/usr/lib",
      "/opt/"
    ],
```

- **Mmap Exec File**: Similar to Process Exec, but for dynamically loaded libraries.
  - **enable**: Turn detection on or off.
  - **remediate**: Turn asynchronous blocking on or off.
  - **inline**: Turn synchronous blocking on or off (remediate must be set to true for inline blocking to take effect)
  - **`allowed_paths`**: List directories where libraries *can* be loaded from.
  - **`excluded_paths`**: List specific library directories that are forbidden.
- **Sensitive File Access**: Flags access to sensitive files like password lists or SSH keys.
  - **enable**: Turn detection on or off.
  - **remediate**: Turn asynchronous blocking on or off.
  - **inline**: Turn synchronous blocking on or off (remediate must be set to true for inline blocking to take effect)
  - **`absolute_paths`**: List exact file paths to watch (e.g., `/etc/shadow`).
  - **`relative_paths`**: List relative file paths to watch (e.g., `/.ssh/id_rsa`).
  - **`allowed_paths`**: List programs that *are* allowed to touch these sensitive files.
- **Process Detection**: Identifies suspicious programs running that may be used for reconnaissance or exploits.
  - **enable**: Turn detection on or off.
  - **remediate**: Turn asynchronous blocking on or off.
  - **inline**: Turn synchronous blocking on or off (remediate must be set to true for inline blocking to take effect)
  - **`suspicious_list_path`**: List paths to programs considered suspicious (e.g., `/nc`, `/wget`).
  - **`exception_list_comm`**: List parent programs that *can* run suspicious processes.
- **Detect SetUGid**: This rule only *detects* when programs that can escalate privileges (like `sudo`) are run or forked. There is no blocking mode for this rule.
  - **enable**: Turn detection on or off.
- **Process Restriction**: Catches specific programs being launched by other restricted programs.

    - **enable**: Turn detection on or off.
    - **remediate**: Turn asynchronous blocking on or off.
    - **inline**: Turn synchronous blocking on or off (remediate must be set to true for inline blocking to take effect)
    - **`process_groups`**: Define which programs can't launch which other programs.
- **Socket Detection**: Watches for suspicious incoming and outgoing network connections.
    - **enable**: Turn detection on or off.
    - **remediate**: Turn asynchronous blocking on or off.
    - **inline**: Turn synchronous blocking on or off (remediate must be set to true for inline blocking to take effect)
    - **allow_list_addr_prefix**: List allowed network address prefixes (e.g., `127.0.0.1`).
    - **`allow_list_comm`**: List programs that are allowed to make network connections.

Appendix

Below is a default JSON policy file:

```
{
  "common": {},
  "kernel_common": {
    "interpreter": {
      "/sh": [
        "c"
      ],
      "/dash": [
        "c"
      ],
      "/bash": [
        "c"
      ],
      "/ksh": [
        "c"
      ],
      "/ash": [
        "c"
      ],
      "/zsh": [
        "c",
        "s"
      ],
      "/python": [
        "c"
      ],
      "/perl": [
        "e",
        "E"
      ]
    },
    "webserver": [
      "httpd",
      "nginx",
      "lighttpd",
      "apache2"
    ],
    "shell": [
      "/sh",
      "/dash",
      "/bash",
      "/ksh",
      "/ash",
      "/zsh",
      "/python",
      "/perl"
    ],
    "nettool": [
```

```
      "nettool": [
        "/nc",
        "/netcat",
        "/netcat-openbsd",
        "/netcat-traditional",
        "/socat"
      ]
    },
    "kernel_integrity": {
      "remediate": false,
      "fileops_strict": false,
      "core_pattern_value": "",
      "modprobe_path_value": "",
      "poweroff_cmd_value": ""
    },
    "timeout": {
      "enable": false,
      "remediate": false
    },
    "userspace_force_nx_stack": {
      "enable": false
    },
    "container_drift": {
      "enable": false,
      "remediate": false,
      "inline": true,
      "container_exception_list": [],
      "pipe_process_exception_list": [
        "docker-entrypoi",
        "containerd-shim",
        "bpftrace"
      ]
    },
    "process_io_bind_sock_control": {
      "enable": true,
      "remediate": true,
      "inline": true,
      "transitive": false,
      "explicit_deny": [
        "/sh",
        "/dash",
        "/bash",
        "/ksh",
        "/ash",
        "/zsh",
        "/python",
        "/perl"
      ],
      "allow_list_path_transitive": [
```

```
      "allow_list_path_transitive": [
        "/usr/lib/systemd/systemd-resolved",
        "/usr/lib/apt/methods/http",
        "/usr/lib/apt/methods/rsh",
        "/usr/lib/apt/methods/mirror",
        "/usr/bin/cri-dockerd"
      ]
    },
    "container_capabilities": {
      "enable": false,
      "remediate": false,
      "caps_denied": [],
      "explicit_allow": []
    },
    "nsenter": {
      "enable": true,
      "remediate": true,
      "inline": true
    },
    "container_socket_protect": {
      "enable": false,
      "remediate": false,
      "inline": true,
      "protected_file_paths": [
        "/var/run/docker.sock"
      ]
    },
    "rogue_container": {
      "enable": false,
      "remediate": false
    },
    "socket_detection": {
      "enable": false,
      "remediate": false,
      "inline": true,
      "allow_list_addr_prefix": [
        "/",
        "127.0.0.1"
      ],
      "allow_list_comm": [
        "sshd"
      ]
    },
    "process_exec": {
      "enable": false,
      "remediate": false,
      "inline": true,
      "allowed_paths": [
        "/bin/",
```

```
        "/sbin/",
        "/usr/bin/",
        "/usr/sbin/",
        "/usr/local/bin/",
        "/usr/local/sbin/",
        "/usr/lib",
        "/opt/"
      ],
      "excluded_paths": []
    },
    "mmap_exec_file": {
      "enable": false,
      "remediate": false,
      "inline": true,
      "allowed_paths": [
        "/lib/",
        "/lib64/",
        "/usr/lib",
        "/usr/local/lib/",
        "/usr/local/lib64/",
        "/opt/"
      ],
      "excluded_paths": []
    },
    "sensitive_file_access": {
      "enable": false,
      "remediate": false,
      "inline": true,
      "absolute_paths": [
        {
          "path": "/etc/passwd",
          "write_only": true
        },
        {
          "path": "/etc/shadow",
          "write_only": false
        }
      ],
      "relative_paths": [
        {
          "path": "/.ssh/id_rsa",
          "write_only": false
        },
        {
          "path": "/.ssh/id_ecdsa",
          "write_only": false
        },
        {
```

```
        "path": "/.ssh/id_ecdsa_sk",
        "write_only": false
      },
      {
        "path": "/.ssh/id_ed25519",
        "write_only": false
      },
      {
        "path": "/.ssh/id_ed25519_sk",
        "write_only": false
      }
    ],
    "allowed_paths": [
      "/usr/sbin/unix_chkpwd",
      "/usr/lib/systemd/systemd-userwork"
    ]
  },
  "process_detection": {
    "enable": false,
    "remediate": false,
    "inline": true,
    "suspicious_list_path": [
      "/nc",
      "/wget",
      "/curl"
    ],
    "exception_list_comm": [
      "setup-policy-ro"
    ]
  },
  "detect_setugid": {
    "enable": false
  },
  "process_restriction": {
    "enable": false,
    "remediate": false,
    "inline": true,
    "process_groups": [
      {
        "restricted": [
          "httpd",
          "nginx",
          "lighttpd",
          "apache2"
        ],
        "executable": [
          "/nc",
          "/netcat",
```

```
            "/netcat-openbsd",
            "/netcat-traditional",
            "/socat"
          ]
        },
        {
          "restricted": [
            "httpd",
            "nginx",
            "lighttpd",
            "apache2"
          ],
          "executable": [
            "/sh",
            "/dash",
            "/bash",
            "/ksh",
            "/ash",
            "/zsh",
            "/python",
            "/perl"
          ]
        }
      ]
    },
    "forced_mem_access": {
      "enable": false,
      "remediate": false,
      "inline": true,
      "prevent_writes_only": true
    },
    "python_sensor": {
      "enable": true,
      "pickle": {
        "enable": true,
        "remediate": false,
        "inline": true,
        "functions_deny_list": [
          "builtins.exec",
          "posix.system",
          "socket.socket"
        ],
        "functions_allow_list": null
      },
      "pathtraversal": {
        "enable": true,
        "remediate": false,
        "inline": true,
```

```
      "deny_list": [
        "/etc/passwd",
        "/etc/shadow"
      ]
    },
    "imports": {
      "enable": true,
      "remediate": false,
      "inline": true,
      "deny_list": [
        ""
      ]
    },
    "execs": {
      "enable": true,
      "remediate": false,
      "deny_list": [
        ""
      ],
      "inline": true,
      "allow_list": [
        ""
      ]
    },
    "loads": {
      "enable": true,
      "remediate": false,
      "inline": true,
      "deny_list_dlsym": [
        ""
      ],
      "deny_list_dlopen": [
        ""
      ],
      "deny_list_load": [
        ""
      ]
    },
    "urllib": {
      "enable": true,
      "remediate": false,
      "inline": true
    },
    "url": {
      "enable": true,
      "remediate": false,
      "inline": true
    },
```

```
    "misc": {
      "enable": true,
      "remediate": false,
      "inline": true
    }
  },
  "java_sensor": {
    "enable": true,
    "java_class_load": {
      "enable": true,
      "remediate": false,
      "inline": true
    },
    "java_file_open": {
      "enable": true,
      "remediate": false,
      "inline": true,
      "deny_list": [
        "/etc/passwd",
        "/etc/shadow"
      ]
    },
    "java_deserialization": {
      "enable": true,
      "remediate": false,
      "inline": true,
      "deny_list": [
        ""
      ],
      "allow_list": [
        ""
      ]
    },
    "java_exec": {
      "enable": true,
      "remediate": false,
      "inline": true
    },
    "java_reflection": {
      "enable": true,
      "remediate": false,
      "inline": true
    },
    "java_misc": {
      "enable": true,
      "remediate": false
    }
  }
}
```

ئ

Testing BlueRock Policies

Once a policy file has been uploaded, you can test the policies using the examples below:

nsenter Protection

Ensure the `nsenter` stanza in the policy file matches the following:

```
    "nsenter": {
      "enable": true,
      "remediate": true,
      "inline": true
    },
```

Then on the BlueRock Node, attempt to move laterally from one namespace to another. You can run the following script on the node to test this capability:


569B

nsenter_test.sh

```
$ ./nsenter_test.sh
Spinning up test webserver container
[+] Running 2/2
 ✔ Network tmp_default  Created
0.2s
 ✔ Container websever   Started
0.7s
[...] Press ENTER to attempt to enter container Namespace with nsenter
nsenter: reassociate to namespace 'ns/pid' failed: Operation not
permitted
Spinning down test webserver container
[+] Running 2/2
 ✔ Container websever   Removed
0.4s
 ✔ Network tmp_default  Removed
```

Then navigate to the CloudWatch log group and use the filter to search for `ERROR` logs:

```
{ $.severity_text = "ERROR" }
```

The following log should appear

```
{
    "severity_number": 17,
    "severity_text": "ERROR",
    "attributes": {
        "event": {
            "description": "'/usr/bin/nsenter' tries to nsenter",
            "name": "nsenter_violation",
            "source_event_id": 420
        },
        "hostid": "bru-host",
        "meta": {
            "cpu_id": 2,
            "name": "namespace_change",
            "source_event_id": 420,
            "type": "remediation"
        }
    }
}
```

Use the `$.attributes.meta.source_event_id` field to collect more information about the incident

```
{ $.attributes.meta.source_event_id = 420 ||
$.attributes.event.source_event_id = 420 }
```

```
{
    "severity_number": 9,
    "severity_text": "INFO",
    "attributes": {
        "comm": "nsenter",
        "context": {
            "cgroup": {
                "cgroup_id": 29086,
                "cgroup_name": "/user.slice/user-1000.slice/
session-15.scope"
            },
            "namespace": {
                "cgroup_ns_inum": 4026532303,
                "ipc_ns_inum": 4026532301,
                "mnt_ns_inum": 4026531841,
                "net_ns_inum": 4026532304,
                "pid_for_children_ns_inum": 4026531836,
                "pid_ns_inum": 4026531836,
                "time_for_children_ns_inum": 4026531834,
                "time_ns_inum": 4026531834,
                "user_ns_inum": 4026531837,
                "uts_ns_inum": 4026532300
            },
            "parent_process": {
                "comm": "sudo",
                "pid": 36257
            },
            "process": {
                "comm": "nsenter",
                "effective_capability": 2199023255551,
                "egid": 0,
                "euid": 0,
                "file_path": "/usr/bin/nsenter",
                "gid": 0,
                "permitted_capability": 2199023255551,
                "pid": 36258,
                "sys_daemon": false,
                "uid": 0
            }
        },
        "hostid": "bru-host",
        "meta": {
            "cpu_id": 2,
            "name": "namespace_change",
            "source_event_id": 420,
            "type": "event"
        },
        "namespace": {
```

```
            "namespace": {
                "cgroup_ns_inum": 4026532303,
                "ipc_ns_inum": 4026532301,
                "mnt_ns_inum": 4026531841,
                "net_ns_inum": 4026532304,
                "pid_for_children_ns_inum": 4026532302,
                "pid_ns_inum": 4026531836,
                "time_for_children_ns_inum": 4026531834,
                "time_ns_inum": 4026531834,
                "user_ns_inum": 4026531837,
                "uts_ns_inum": 4026532300
            },
            "pid": 36258
        }
    }
```


Capability Protection

Review the following stanza

```
    "container_capabilities": {
      "enable": true,
      "remediate": true,
      "caps_denied": ["CAP_CHOWN"],
      "explicit_allow": []
    },
```

This policy denies every container the ability to use the `CAP_CHOWN` capability, blocking any change of owners for files in the container. We will spin up a container and attempt to alter the permissions of a file. Even though the container has the permission to use `CAP_CHOWN`, BlueRock blocks the attempt

```
#!/bin/bash
echo "Spinning up capability container"
COMPOSE_FILE=$(mktemp)

cat > "$COMPOSE_FILE" <<EOF
services:
  trusted_file_access:
    image: alpine
    container_name: cap_test
    cap_add:
      - CHOWN
      - DAC_OVERRIDE
      - DAC_READ_SEARCH
      - FOWNER
      - FSETID
    command: sh -c "sleep infinity"
EOF
docker-compose -f "$COMPOSE_FILE" up -d
read -p "[...] Press ENTER to attempt to enter use a prohibited
container capability"
docker exec cap_test sh -c "touch /tmp/test && chown 1000:1000 /tmp/
test"
echo "Spinning down cap test container"
docker-compose -f "$COMPOSE_FILE" down
rm -f "$COMPOSE_FILE"
```

Copy the above script onto a BlueRock node with the policy stanza above and run it to see the following

```
./cap_test.sh
Spinning up capability container
[+] Running 2/2
 ✔ Network tmp_default  Created
0.2s
 ✔ Container cap_test   Started
0.4s
[...] Press ENTER to attempt to enter use a prohibited container
capability
Spinning down cap test container
[+] Running 2/2
 ✔ Container cap_test   Removed
10.3s
 ✔ Network tmp_default  Removed
```

Then using the filters above you can find the following logs in CloudWatch


```
{
    "body": {
        "event.domain": "gyro",
        "event.id": 274,
        "event.name": "capable_violation",
        "event.type": "remediation",
        "hostid": "bru-single",
        "meta": {
            "description": "Container
'f2129b2302e09398476e6ed5d79aa87598e344535fbcb71b106603bd6fbfe46f' is
trying to use capability 'CAP_CHOWN'",
            "name": "capable_violation",
            "sensor_id": 8825,
            "source_event_id": 274
        }
    },
    "severity_number": 17,
    "severity_text": "ERROR",
    "attributes": {
        "event.domain": "gyro",
        "event.id": 274,
        "event.name": "capable_violation",
        "event.type": "remediation",
        "hostid": "bru-single",
        "meta": {
            "description": "Container
'f2129b2302e09398476e6ed5d79aa87598e344535fbcb71b106603bd6fbfe46f' is
trying to use capability 'CAP_CHOWN'",
            "name": "capable_violation",
            "sensor_id": 8825,
            "source_event_id": 274
        }
    }
}
```


And the additional context

```
{
    "body": {
        "aux": {
            "container_metadata": {
                "container":
"f2129b2302e09398476e6ed5d79aa87598e344535fbcb71b106603bd6fbfe46f"
            }
        },
        "cap_name": "CAP_CHOWN",
        "cap_num": 0,
        "context": {
            "cgroup": {
                "cgroup_id": 13892,
                "cgroup_name": "/system.slice/docker-
f2129b2302e09398476e6ed5d79aa87598e344535fbcb71b106603bd6fbfe46f.scope"
            },
            "namespace": {
                "cgroup_ns_inum": 4026532368,
                "ipc_ns_inum": 4026532366,
                "mnt_ns_inum": 4026532364,
                "net_ns_inum": 4026532369,
                "pid_for_children_ns_inum": 4026532367,
                "pid_ns_inum": 4026532367,
                "time_for_children_ns_inum": 4026531834,
                "time_ns_inum": 4026531834,
                "user_ns_inum": 4026531837,
                "uts_ns_inum": 4026532365
            },
            "parent_process": {
                "comm": "containerd-shim",
                "pid": 11885
            },
            "process": {
                "comm": "chown",
                "effective_capability": 2818844159,
                "egid": 0,
                "euid": 0,
                "file_path": "/bin/busybox",
                "gid": 0,
                "permitted_capability": 2818844159,
                "pid": 11960,
                "sys_daemon": false,
                "uid": 0
            }
        },
        "event.domain": "sensor",
        "event.id": 274,
        "event.name": "capable"
```

```
        "event.name": "capable",
        "event.type": "event",
        "hostid": "bru-single",
        "meta": {
            "domain": "sensor",
            "name": "capable",
            "sensor_id": 8825,
            "source_event_id": 274,
            "type": "event"
        }
    },
    "severity_number": 9,
    "severity_text": "INFO",
    "attributes": {
        "aux": {
            "container_metadata": {
                "container":
"f2129b2302e09398476e6ed5d79aa87598e344535fbcb71b106603bd6fbfe46f"
            }
        },
        "cap_name": "CAP_CHOWN",
        "cap_num": 0,
        "context": {
            "cgroup": {
                "cgroup_id": 13892,
                "cgroup_name": "/system.slice/docker-
f2129b2302e09398476e6ed5d79aa87598e344535fbcb71b106603bd6fbfe46f.scope"
            },
            "namespace": {
                "cgroup_ns_inum": 4026532368,
                "ipc_ns_inum": 4026532366,
                "mnt_ns_inum": 4026532364,
                "net_ns_inum": 4026532369,
                "pid_for_children_ns_inum": 4026532367,
                "pid_ns_inum": 4026532367,
                "time_for_children_ns_inum": 4026531834,
                "time_ns_inum": 4026531834,
                "user_ns_inum": 4026531837,
                "uts_ns_inum": 4026532365
            },
            "parent_process": {
                "comm": "containerd-shim",
                "pid": 11885
            },
            "process": {
                "comm": "chown",
                "effective_capability": 2818844159,
                "egid": 0,
                "euid": 0
```

```
                "file_path": "/bin/busybox",
                "gid": 0,
                "permitted_capability": 2818844159,
                "pid": 11960,
                "sys_daemon": false,
                "uid": 0
            }
        },
        "event.domain": "sensor",
        "event.id": 274,
        "event.name": "capable",
        "event.type": "event",
        "hostid": "bru-single",
        "meta": {
            "domain": "sensor",
            "name": "capable",
            "sensor_id": 8825,
            "source_event_id": 274,
            "type": "event"
        }
    }
}
```