IAM PrivEsc Labs

Table of Contents

[Introduction 1](#_Toc52922480)

[Prerequisites 1](#_Toc52922481)

[Environment Setup 1](#_Toc52922482)

[Time to start the ‘real’ part of the Lab! 5](#_Toc52922483)

[1. Creating a new Policy Version 5](#_Toc52922484)

[2. Setting the default policy version to an existing version 7](#_Toc52922485)

[3. Creating an EC2 instance with an existing instance profile 9](#_Toc52922486)

[Alternate Installation and Runtime Methods using Docker 9](#_Toc52922487)

[Use Amazon Linux interactively (best alternate option) 9](#_Toc52922488)

[Use the image of your choice (intermediate option) 9](#_Toc52922489)

[Use the AWS-provided container (worst option) 9](#_Toc52922490)

[References/Links/Resources 9](#_Toc52922491)

# Introduction

* Many resources just show “best practice” or similar; very few tools demonstrate specific combinations of scenarios in which priv esc is possible
* This is a hands-on walkthrough of how to do just that.

# Prerequisites

* Must have your own AWS account to use for this
* Must have sufficient permissions on your local machine to install the AWS CLI tools (or another way to run this, such as from a docker container)
  + Linux users: you will need the following packages if not already installed: less, unzip, ssh, ssh-keygen
* Runtime cost estimate: $0.02 (two cents US)
  + t2.micro costs approx. $0.0116 (that’s just over one cent per hour, USD). This lab may involve using 1 hour of this compute cost. Users wishing to not do this may opt to not participate in this portion.
  + Lambda costs approximately $0.20 per million requests; this lab may involve up to 50 requests. While this is not possible to bill (1/1000 of a cent), AWS may require billing information to be configured to run the Lambda section. Users wishing to not do this may opt to not participate in this portion.

# Environment Setup

* Download and install the AWS CLI on your machine. Installer links per OS (Windows, MacOS, Linux, Amazon Linux) can be found on the top right of the page.  
  <https://aws.amazon.com/cli/>
* Log into your AWS account
* Create an IAM user called labuser1
  + Type IAM in the search bar or navigate to IAM
  + Click “Users”
  + Click the blue “Add User” button
  + In the **User name\*** text box, enter  
    labuser1
  + Under Select AWS access type, note the 2 types of access and their constraints. We will NOT allow this user console access, so **select the checkbox for Programmatic Access** (only).
  + At the bottom right of the page, click the blue **Next: Permissions** button
  + Your user defaults to having no permissions at all; we will now create a custom policy for this user and will assign that policy to the user.
    - There are 3 ‘card’ style buttons at the top. Click the one that says “Attach Policies directly”
    - Notice that this now created a button just under this row of cards that says “Create Policy”. **Click that button**
    - This will open a new tab, and you’ll need to now create a new policy to define the permissions that are able to be assigned. Let’s explore this in the visual editor so that you can see what all sorts of options are around. In **Service**, select **IAM**.
      * In actions, under the Access Level, select the checkbox next to **List** to automatically select all 30 List actions.
        + In JSON format, this would be adding  
          iam:List\*
      * In Actions, Expand the Permissions Management section under Access Level. Select the checkbox next to **CreatePolicyVersion.**
        + Do NOT select “CreatePolicy”
      * In the Resources section, select the radio button next to “All Resources”
        + It may be helpful to expand this to see what all is being added. For this lab, we are adding everything, but this is not how things would be at work.
      * In the bottom right of the window, click the blue **Review policy** button
      * Give your policy a memorable name so that it can be easily removed later. For this example, I will use:  
        aaa-iam-escalation-lab  
          
        I \*HIGHLY\* recommend using **aaa** as a prefix in this lab. The reason why will become more clear later in this lab.
      * In the bottom right, click the blue Create Policy button
      * You should be redirected back to the IAM Policies page, and most likely this policy is at the top of the list, as shown below. **IF YOUR POLICY IS NOT PRESENT, TROUBLESHOOT THEN ASK YOUR INSTRUCTOR.**Graphical user interface, text, application

        Description automatically generated
      * You may now close the tab that this was in; you should return to the Add User tab.
    - Note that your new policy is not yet visible. Refresh the page contents by clicking the refresh button () on the right side of the page. You can then select your policy as shown below:   
      Graphical user interface, text, application

      Description automatically generated
    - Click the blue **Next: Tags** button
    - Click the blue **Next: Review** button
    - Click the blue **Create User** button
    - **IMPORTANT:** This is the **ONLY** time you will have access to this key, and this key has the ability to escalate privileges in your account. BE SURE TO SAVE THIS IN A SAFE PLACE!!!
      * Click the Download CSV button and select a place to store this credential pair.
      * Click the Close button to return to the IAM Users page
      * Note that your new user should show “None” on the access key age, Password age, Last Activity, and “Not Enabled” on the MFA sections.
        + Remember: Use MFA wherever possible! This is a key way to protect yourself!
* See the IAM Policy in JSON Format
  + In IAM -> Policies, click your new policy. By default, it will show in the Policy Summary view
  + Click the **{ } JSON** button to view your policy in JSON format. *This is the format that is most commonly seen, used, and worked with in the company. You should familiarize yourself with this syntax and structure.*
    - Looking through this, there are a few notable things to point out:
      * Effect: Allow – we are creating permissions to ALLOW things
        + Recall that there is an implicit deny in AWS; if it’s not specified, generally it is a default deny all (we still prefer explicit deny wherever practicable)
        + These read like a firewall ruleset: top to bottom, and inheritance matters
        + AWS provides an excellent Policy Evaluation flowchart. I personally have this printed & laminated for easy troubleshooting  
          <https://docs.aws.amazon.com/IAM/latest/UserGuide/images/PolicyEvaluationHorizontal.png>

Source Page with more description & details:   
<https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html>

* + - * Lots of List actions (30), there’s the CreatePolicyVersion that we made. There’s also these two:
        + **iam:GetLoginProfile** – retrieves the user name and password-creation date for a specified IAM user. Seems like this might be related to password rotations, BUT – no, that’s handled another way. This is added automatically by the Policy Editor because the iam:ListAccessKeys permission requires it; that command outputs the key age, as shown below:   
          Text

          Description automatically generated
        + **iam:GetAccountSummary** – Retrieves information about IAM entity usage & quotas in the account. Note that when given all the list actions, this same information seems like it could be intuited, BUT – this is a prerequisite permission for iam:ListVirtualMFADevices. Since we added all the List actions, this was added on for us by the Policy Editor.
        + **These are great examples of AWS helpfully adding things that you may not have intended**. Be aware of everything it’s adding and everything you need!
* Configure your AWS CLI Environment
  + On your machine, open your favorite terminal, and type “aws configure”
    - If you have installed the AWS CLI correctly this should prompt you with the below. Copy/paste the Access Key and Secret Access Key from your CSV download earlier.
      * AWS Access Key ID [None]:
      * AWS Secret Access Key ID [None]:
      * Default region name [None]: NOTE: select whichever region you use most
      * Default output format [None]: NOTE: you may leave this blank (hit enter)
    - Example   
      A picture containing clock, object, green, street

      Description automatically generated
  + Validate your user’s permissions and the CLI configuration by running:  
    aws iam list-users  
    Graphical user interface, application

    Description automatically generated
  + Validate that you can **NOT** see output from s3 by running a command to list s3 buckets:   
    aws s3 ls  
      
    You should see something like this:   
    A close up of a clock

    Description automatically generated

# Time to start the ‘real’ part of the Lab!

## Creating a new Policy Version

In part 1, you will see how an attacker with the **iam:CreatePolicyVersion permission** (this was the extra one we added in addition to the List\* actions earlier) can create a new version of an IAM policy that they have access to. This allows them to define their own custom permissions. When creating a new policy version, it needs to be set as the default version to take effect, which you would think would require the iam:SetDefaultPolicyVersion permission, but when creating a new policy version, it is possible to include a flag (--set-as-default) that will automatically create it as the new default version. That flag does not require the iam:SetDefaultPolicyVersion permission to use, and this is how this privilege escalation attack works.

* In your terminal, run:   
  vi policy.json
* Paste the following contents including { } into that file, then save and quit VI:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "\*",

"Resource": "\*"

}

]

}

* Now run:   
  aws iam list-policies
* Note that if you named your policy with aaa- at the beginning, this is very likely going to be the first policy – that’s good, because there are 500+ default policies. *Hopefully you followed the advice on naming!*
* Copy the Arn of your policy (everything between the quotes)
* Press q to exit out of the “less” paginated output
* **ATTACK!** Run the following command (NOTE: double dashes on front of flags)  
  aws iam create-policy-version --policy-arn INSERT\_YOUR\_ARN\_HERE --policy-document file://policy.json --set-as-default  
    
  Graphical user interface, text, application, website

  Description automatically generated
* Try something you didn’t have before, such as the one we ran earlier:   
  aws s3 ls
  + Even if you don’t have any buckets, this should show something other than “denied”.
* Compare versions:
  + aws iam get-policy-version --policy-arn YOUR\_POLICY\_ARN --version-id v1
  + aws iam get-policy-version --policy-arn YOUR\_POLICY\_ARN --version-id v2

\*\*\* VALIDATION IN THE GUI\*\*\*

* Back in your web session, go to IAM -> Policies -> click on YOUR POLICY (e.g.: aaa-iam-escalation-lab)
  + If this is still here from before, click “Policies” and go back into it to pull a fresh update.
  + It may default to the JSON editor. If not, Click JSON and Note that the permissions are allow \* on \*
  + Click the Policy Summary, and note that the access level is now FULL ACCESS on ALL RESOURCES, all the way down the page

**SUMMARY:** You have now escalated from a seemingly low-level read account with one extra permission to full admin. Congrats!

## Setting the default policy version to an existing version

Sometimes there’s a policy that’s made and later hardened. This lab will demonstrate how these old over-permissioned versions can be dangerous to your environment. Let’s start off by editing that user’s permissions using the same privilege escalation technique (though in this case we’re de-escalating it).

* In your terminal,   
  aws iam get-policy-version --policy-arn YOUR\_POLICY\_ARN --version-id v1
* Copy the Action block, then change the local policy to have those, removing the CreatePolicyVersion, and adding iam:SetDefaultPolicyVersion (copy/paste-able version below)
* Run:  
  vi 2policy.json <- note: in order to keep the policies around, I’m using a second file here

Copy/Paste-able Policy Document for 2policy.json:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"iam:ListRoleTags",

"iam:ListServerCertificates",

"iam:ListPoliciesGrantingServiceAccess",

"iam:ListServiceSpecificCredentials",

"iam:ListMFADevices",

"iam:ListSigningCertificates",

"iam:ListVirtualMFADevices",

"iam:ListInstanceProfilesForRole",

"iam:ListSSHPublicKeys",

"iam:ListAttachedRolePolicies",

"iam:ListAttachedUserPolicies",

"iam:ListAttachedGroupPolicies",

"iam:ListRolePolicies",

"iam:ListAccessKeys",

"iam:ListPolicies",

"iam:ListSAMLProviders",

"iam:ListGroupPolicies",

"iam:ListEntitiesForPolicy",

"iam:ListRoles",

"iam:ListUserPolicies",

"iam:ListInstanceProfiles",

"iam:ListPolicyVersions",

"iam:ListOpenIDConnectProviders",

"iam:ListGroupsForUser",

"iam:ListAccountAliases",

"iam:ListUsers",

"iam:ListGroups",

"iam:GetLoginProfile",

"iam:ListUserTags",

"iam:GetAccountSummary",

"iam:SetDefaultPolicyVersion"

],

"Resource": "\*"

}

]

}

Now we can simply run the priv-esc again, noting the new file name of 2policy.json.  
aws iam create-policy-version --policy-arn YOUR\_POLICY\_ARN --policy-document file://2policy.json --set-as-default

**TAKING STOCK:** Right now, we’re back to a relatively low-permissioned that has List\* on IAM, a couple get statements to support those list ones, and the SetDefaultPolicyVersion permission. We can actually validate that we lost some permissions by re-running the get-policy-version command from earlier, incrementing the version now to v3:  
aws iam get-policy-version --policy-arn YOUR\_POLICY\_ARN --version-id v1

Note that you no longer have permission to run this.

**Here’s the thing**: we know that there’s a policy version that has more permissions. Typically this is in low version numbers, as people tend to create one either overly permissioned or blank, then they iterate on it. Sometimes people get frustrated or flustered, and after 5, 10, 15 or so iterations not working they’ll give up and assign way too many permissions. This is what the attacker will be going for. In this case, we’ll go straight for the gold. *If you have more versions than shown here, use the version # of the one with full permissions in the next command*.

**ATTACK!** Run the following command (NOTE: double dashes on front of flags)

aws iam set-default-policy-version --policy-arn YOUR\_POLICY\_ARN --version-id v2

Validate your abilities with the below commands:

aws iam get-policy-version --policy-arn YOUR\_POLICY\_ARN --version-id v2

aws s3 ls

\*\*\* VALIDATION IN THE GUI\*\*\*

* Back in your web session, go to IAM -> Policies -> click on YOUR POLICY (e.g.: aaa-iam-escalation-lab)
  + If this is still here from before, click “Policies” and go back into it to pull a fresh update.
  + It may default to the JSON editor. If not, Click JSON and Note that the permissions are allow \* on \*
  + Click the Policy Summary, and note that the access level is now FULL ACCESS on ALL RESOURCES, all the way down the page

**SUMMARY:** You have now escalated from a seemingly low-level read account with one extra permission to full admin. Congrats! \*\*\*NOTE: In this case it happens to be full admin, but this attack is limited to whatever the most permissive version of a given policy is. Privilege Escalation attacks often are chained from one to another though, so this has serious potential if not properly secured. *Remember, up until that one-liner, this looked just like an engineer trying to get their job done.* Key takeaway: Even inactive policies can carry risk.

## Creating an EC2 instance with an existing instance profile

**Description**

An attacker with the iam:PassRole and ec2:RunInstances permissions can create a new EC2 instance that they will have operating system access to and pass an existing EC2 instance profile/role to it. They can then login to the instance and request the associated AWS keys from the EC2 instance meta data, which gives them access to all the permissions that the associated instance profile/role has.

**Required Permission(s)**

iam:PassRole

ec2:RunInstances

**Potential Impact**

This attack would give an attacker access to the set of permissions that the instance profile/role has, which again could range from no privilege escalation to full administrator access of the AWS account.

**Instructions:**

* This is the scenario format we’ll start seeing in the labs, as the later labs will not have a full walkthrough; some self-discovery is important.
* Navigate to AWS -> EC2 -> Instances
* In the upper right, click the orange button which says “Launch Instances”
* The AMI used for this lab is free tier eligible and very lightweight – it’s the Amazon Linux 2 AMI provided by AWS. This course was written using the following AMI ID on 2020-OCT-07, though the latest version (if existing) should be just fine:   
  ami-03657b56516ab7912  
  ->Click the blue Select button to the right side of this.
* Leave the instance type at the default (t2.micro, which is free tier eligible). In the lower right, click the gray button which says “**Next: Configure Instance Details**”
* On the Configure Instance Details screen, about halfway down is an option for IAM role. **This is the part which this lab intends to exploit**. It is common for EC2 instances to have roles assigned (hence the option here). Some examples are execution roles for Elastic Bean Stalk. In fact, this is so common that the specific escalation we are going to exploit *and all the permissions needed* is in [this AWS IAM role for EC2 User Guide (link).](https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/iam-roles-for-amazon-ec2.html) Scroll down to the section called **Granting an IAM user permission to pass an IAM role to an** instance and note that the 2 permissions we specified in the description above are both in this example. Just under their IAM policy code block there’s a brief line mentioning that you should consider what permissions your users need, but it doesn’t notate that this policy carries with it some inherent risk.
* We are going to create a new role, so back on that Configure instance details screen, click “Create new IAM role”. This will open a new AWS Console Tab to the IAM Roles page.
  + In the middle-left of the screen, click the blue button labeled “Create Role”
  + Leave the ‘cards’ at the top selected on AWS Service (defaults to there from this flow). Select EC2 in the list below, then in the bottom right of the window click the blue button labeled “Next: Permissions”
  + Select the checkbox next to the policy we made earlier—in my case that’s aaa-iam-escalation-lab, and it’s at the top of the list. Then in the bottom right of the window click the blue button labeled “Next: Tags”.
  + We don’t need to add tags right now, so in the bottom right of the window click the blue button labeled “Next: Review”
  + Give the role a name. Much like before, I’m going to prefix with aaa- to ensure it’s at the top of any output. I’ll use:  
    aaa-iam-escalation-lab-ec2-role
  + Finally, in the bottom right of the window click the blue button labeled “Create role”, then you can close the IAM tab.
* Back in the Configure Instance Details window, you’ll need to refresh the dropdown list next to the IAM Role selector, then select the role you just created (aaa-iam-escalation-lab-ec2-role), and in the bottom right of the window click the blue button labeled “Next: Add Storage”.
* In the bottom right of the window click the blue button labeled “Next: Add Tags”.
* In the bottom right of the window click the blue button labeled “Next: Configure Security Group”.
  + At this point, I highly recommend going to ipchicken.com and getting your public IP address. This EC2 instance will be created with the ability to take over your account, so it would be a good idea to limit its ability to accept incoming traffic to just your home IP.
  + Enter that IP address returned from ipchicken.com into the Source field, followed by /32. For example (this is a made-up IP – use your public IP) :   
    45.56.123.200/32  
      
    Enter a description, such as “SSH to vulnerable EC2 instance from my home only”
  + I recommend changing the security group name at the top to something descriptive, such as:   
    aaa-iam-escalation-lab-sg
  + This limits inbound connections to just your public IP and helps you easily spot what the security group is by description.
* In the bottom right of the window click the blue button labeled “Review and Launch”, then review everything, and again in the bottom right, click “Launch”
* You will be prompted to select your key pair. If you already have one you’d like to use, you can do that; however, I recommend making a new key pair just for this lab so that all traces of our activity today can be wiped out later.
  + In the first dropdown, select “Create a new key pair”
  + In the textbox, enter a key pair name. I’m using aaa-iam-escalation-keypair
  + Click Download Key Pair, and keep this safe/note where it’s saved. This is the private key which is used for passwordless authentication to the EC2 host you’ve just created.
    - You will NOT be able to access this key pair again from AWS, so please take note of where it’s stored.
  + Click the now-active Launch Instances button once you have that saved.
  + Click the blue View Instances button to return to the EC2 Instances page
* While your instance initializes, we will configure your local machine to create an SSH connection.
  + Select the checkbox next to your new EC2 instance, then in the upper right click the white button labeled **Actions**. In the dropdown, select “Connect”.
  + The new window should show 3 ‘tabs’ under the Connect to Instance heading. Click the **SSH Client** one.
  + Follow the instructions on screen to connect to your EC2 host.
    - Docker users: note the additional step in the alternate runtime instructions->Amazon Linux->EC2 Lab Note section.
* Now that you’ve connected, let’s explore what sorts of abilities you have. One common attack scenario is to query the EC2 Metadata, which uses the same address for all EC2 hosts, so it’s a very repeatable and scalable attack.
  + Enter the below into your EC2 SSH session:  
    TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600"` && curl -H "X-aws-ec2-metadata-token: $TOKEN" -v http://169.254.169.254/latest/meta-data/
    - To break down this command, the first part sets a variable called TOKEN, and it is defined by a curl command to fetch the metadata API token from the localhost. Then (&&) it runs a curl command, specifying a header with that token, to the metadata service URI.
    - You should see a list of things returned. Try adding those to the end of the command (note: after the first time you don’t need the first half of the above command; however, there’s no harm in this case just hitting the up arrow and adding params to the end. Some examples to try:   
      \*\*/latest/meta-data/instance-type   
      -> Should return t2.Micro. This helps attackers know how they might be able to use a given instance, as well as how careful they may need to be about introducing performance issues/getting caught.  
      \*\*/latest/meta-data/iam/  
      \*\*/latest/meta-data/iam/info  
      ->This last one will show you the associated Role ARN, so as you can see this can be useful later, as these often are named in ways that might tip you off as to what they may be able to do.
    - Note that it’s pretty common for EC2 instances to have permissions to read/write s3 bucket contents. In this instance we didn’t configure that, but there’s a common way to grab credential pairs from the metadataservice using the above structure, ending in iam/security-credentials/s3access
  + Let’s try something else. Rather than JUST using the metadata service, let’s see what permissions we have attached through the AWS CLI in the EC2 instance
    - Run the following commands and check out the output  
      aws s3 ls  
      aws iam list-users  
      aws sts get-caller-identity  
      ^ this one will show you ‘who you are’. Note that your role is able to do a lot so far! Go ahead and copy the role name (aaa-iam-escalation-lab-ec2-role)
  + Now we’ll find out the scope of our powers.
    - Enter the below:  
      aws iam list-attached-role-policies --role-name YOUR\_ROLE\_NAME
    - Now you should see your aaa-iam-escalation-lab policy and its ARN. Let’s investigate its permissions:   
      aws iam get-policy --policy-arn YOUR\_POLICY\_ARN
    - This will tell you what the default version being applied is. In my case, it’s v2, but use whatever yours shows in the following command:  
      aws iam get-policy-version --policy-arn YOUR\_POLICY\_ARN --version-id v2
* **Success!** You should now see that the EC2 instance you (the attacker) has managed to get OS access to has full administrative permissions on your entire AWS account (Action \*, resource \*, effect allow, is default version).

# Lab Cleanup

This will walk you through cleaning up only the parts described above; anything you did outside of that or keep working on may still exist after this cleanup. This assumes you named things as described above; if not, find the one you created.

1. EC2:
2. IAM: Delete the following:
   1. Policies -> aaa-iam-escalation lab
   2. Roles -> aaa-iam-escalation-lab-ec2-role
   3. Users -> labuser1
   4. Groups -> If you made any groups, delete those here.
3. IAM: Enable MFA  
   Not strictly part of the lab, but you REALLY should have MFA on your accounts.

# Alternate Installation and Runtime Methods using Docker

You may have AWS configurations already set up on your machine that you do not wish to tamper with. If this is the case, and if you are feeling more advanced (**not recommended** for most users in this course), you can use one of the below methods to do the lab without tampering with existing configs. Docker would of course be a prerequisite, and this lab does not offer any troubleshooting support for this method or Docker in general.

Reference info can be found here:   
<https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-docker.html>

## Use Amazon Linux interactively (best alternate option)

* Best: because you can keep everything self-contained here, it’s really lightweight, and has most of the tools you require for this lab.
* Steps:
  + docker run -it amazonlinux /bin/bash
  + curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"
  + yum install unzip less openssh-clients -y
  + ssh-keygen (then just hit enter for all the prompts)
  + unzip awscliv2.zip
  + ./aws/install
  + Validate: run “aws --version”
* EC2 Lab Note: When we create the private key, save it on your endpoint, then cat the file (or open in Notepad if on Windows) to get its contents. Copy/paste that into a new file with the same name in your container’s default directory. E.g.:
  + [in container, default dir]   
    touch hackLabKeyPair.pem  
    vi hackLabKeyPair.pem
  + [on endpoint]  
    cat <path to download of keypair>  
    … copy the output, paste into the vi session…
  + [in container, VI session]  
    write/quit
  + You’re now ready to run the EC2 connection commands that are auto-generated from the console!

## Use the image of your choice (intermediate option)

* Intermediate: because you are on self-support mode, but can isolate your pre-existing environment from the lab.

## Use the AWS-provided container (worst option)

* Worst: because you still need the .aws directory on your machine and you have to share that with the container to successfully use aws configure. If you’re using Docker for this lab to try to isolate & leave your pre-existing environment alone, this does NOT meet that need.
* Method: The official amazon image can be instantiated, run, and removed per-command using the below example:   
  docker run --rm -it amazon/aws-cli command

# References/Links/Resources

This lab uses Rhino Security Labs work and publications.

* [Their GitHub](https://github.com/rhinosecuritylabs)
* [The Lab Series we’re using today – AWS-IAM-Privilege-Escalation](https://github.com/RhinoSecurityLabs/AWS-IAM-Privilege-Escalation)