Scenario overview

Christophe • August 27, 2023

A few lessons back, we created an IAM policy that looked like this:

```
{
        "Version": "2012-10-17",
        "Statement": [
                {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "iam:Get*",
                "iam:List*",
                "iam:Put*",
                "iam:SimulateCustomPolicy",
                "iam:SimulatePrincipalPolicy"
            ],
            "Resource": "*"
        }
        ]
}

```

While at first glance this policy may seem harmless, there's a critical issue. To figure out what that is, we first need to understand how IAM policies work. If you're already very familiar with IAM policies, skip forward a little bit, but otherwise this is important.

Anatomy of an IAM policy

To explain how IAM policies work, let's pull up this IAM cheat sheet that I created.
And let's go from top to bottom.

- **Version** – this refers to the version of the policy element, *not the policy version itself.* It defines the version of the policy language. 2012-10-17 is the latest version and should be used to access the latest features
- **Statement** – this is the policy statement itself, and where you add what's allowed or denied using Effects, Actions, and Resources
- **Sid –** the sid is simply the statement id, where each sid needs to be unique within that specific policy
- **Effect**: "Allow" – allows the specified action(s) against the specified resource(s)
- **Effect**: " Deny" – explicit deny which denies the specified action(s) against the specified resource(s)
- **No effect (implicit deny)** – By default, AWS IAM assumes actions aren't allowed for security reasons. This means that if you don't specify an allow statement, it will be implicitly denied. This is very different from Explicit denies ("Effect": "Deny") which overrules any Allow
- **Action** – The action to be allowed or denied. This can a single action or an array of actions
- **Resource** – This is the resource(s) for which the effect and action(s) will be applied. For example, it can be ec2 resources, s3 resources, roles, users, etc...
- **Condition** – Specific conditions for when a policy is in effect. This is an optional element and it can take it a number of different condition operators (in this example: ForAllValues:StringNotLike, condition keys (in this example: aws:TagKeys, and condition values (in this example, Production)
    - We don't have a condition in our example policy but it is important to know

The major issue with our example policy

So back to our prior example policy:

```
{
        "Version": "2012-10-17",
        "Statement": [
                {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "iam:Get*",
                "iam:List*",
                "iam:Put*",
                "iam:SimulateCustomPolicy",
                "iam:SimulatePrincipalPolicy"
            ],
            "Resource": "*"
        }
        ]
}

```

When we have an action of `iam:Get*`, that means whoever or whatever has this policy assigned to it is allowed to run all IAM actions that start with `Get`, which is used to retrieve data.

When we have an action of `iam:Put*`, we have access to all IAM actions that start with `Put`, which is generally a `write` action, meaning that it lets us change things.

The problem is that this includes the action of `PutUserPolicy` which "Adds or updates an inline policy document that is embedded in the specified IAM user."

In other words, whoever or whatever has access to this policy is able to create an inline admin-level policy, and then `PutUserPolicy` to our own user in order to give our user admin privileges.

So even if that was *the only* permission that our user was given, we would be able to escalate our access to become admins of that AWS account. Crazy, right?

That's going to be the scenario that we exploit in this section of the course, so go ahead and complete this lesson, and let's do it.