Damn Vulnerable AWS API

sdmay24-11

Attack Path 1: Ashler Benda, Karthik Kasarabada, Andrew Bowen

Attack Path 2: Garrett Arp, Ahmed Nasereddin, Ayo Ogunsola, Ethan Douglass

Industry Client: Jon Schnell on behalf of RSM US

Faculty Advisor: Julie Rursch

The Problem

- Over 90% of organizations in 2021 were using the cloud for some IT functionality
  (Source: O-Reilley)
- Threat actors are mastering exploitation of common oversights in cloud security.
  (Source: Palo-Alto)
- Personal data breaches were the 2nd most common Cyber Crime reported in 2022
  (Source: FBI Internet Crime Report)
- Overall estimated losses due to *reported* Cyber Crimes in the last five years was $27.6 Billion
  (Source: FBI Internet Crime Report)
- Common free to access cyber security content providers, such as HTB and TryHackMe, have little to no cloud focused training exercises that are available for free

Who cares? What difference will it make?

Users

- IT Administrators
- Software Architects
- Cybersecurity Students
- Risk Consultants
- Application Developers

Use Cases

- Testing and Development
- Security
- Education

Project Requirements

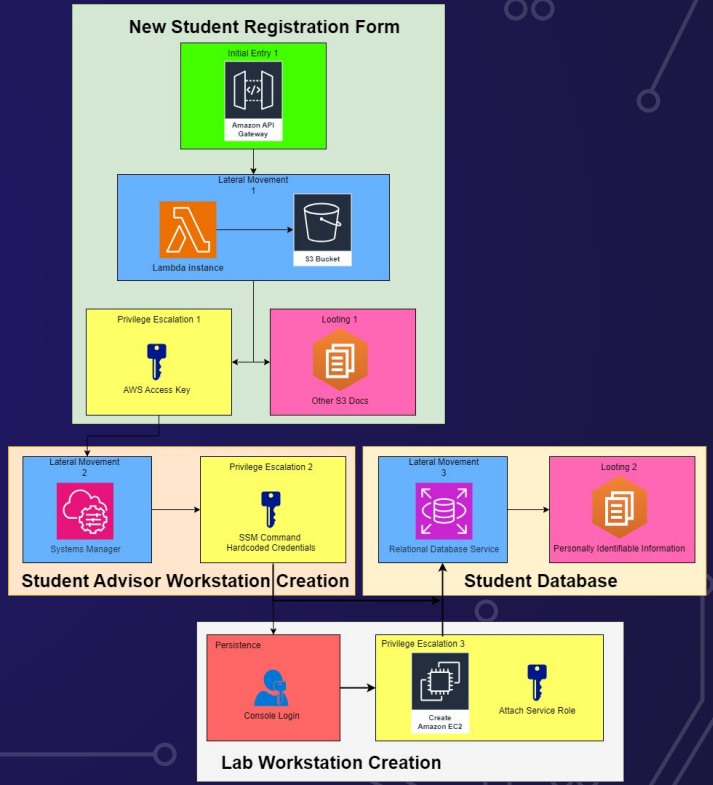
- **Functional Requirements**
  - Incorporates common AWS-specific vulnerabilities and misconfigurations
  - Vulnerabilities should be actively exploitable
  - Attack path includes Non-Volatile persistence in the AWS account
- **Resource Constraints**
  - Utilize AWS CloudFormation for consolidated/static resource configuration and distribution
  - AWS API Gateway should be used as an interface between a user and other AWS resources
  - Utilize AWS Identity and Access Management for resource permissions
  - Cloud resource usage should be minimal, if not all in the free tier
- **Qualitative Requirements**
  - Identity Management roles and policies should reflect professional roles and use cases

Final Design - Attack Path 1

Attack Path based on University network scenario

Common AWS services
- API Gateway
- Lambda Function
- EC2 Instances
- S3 Bucket
- RDS

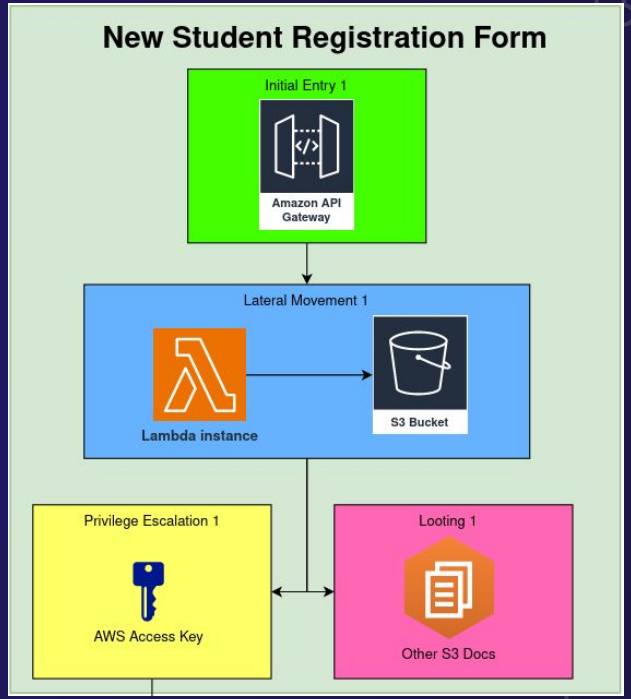
Common mistakes
- Reusing passwords across services
- Not properly deleting old services/tools
- Hardcoding passwords
- Incorrect read/write permissions



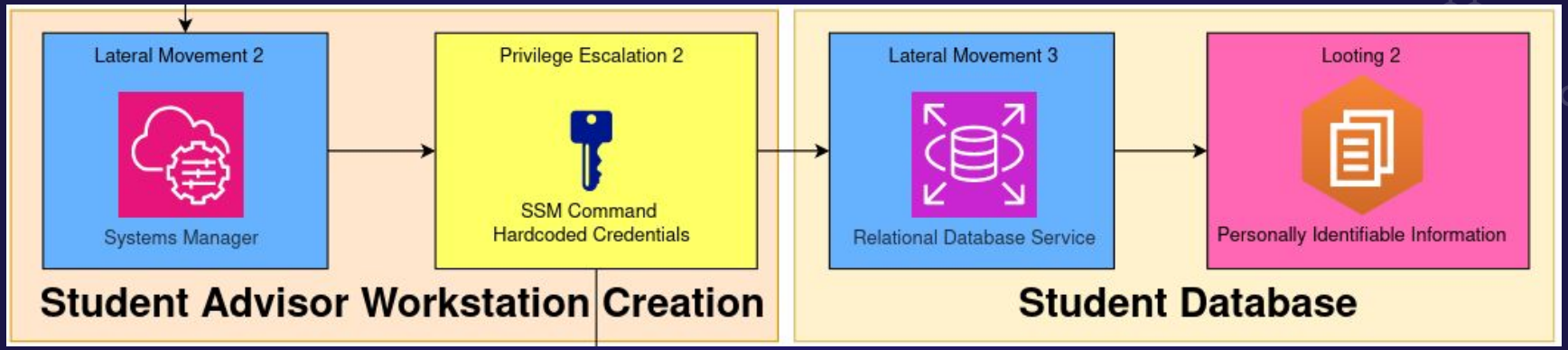
Attack Path 1

New Student Registration Form

- Misconfigured S3 Bucket Permissions
  - Accessible through Lambda Function and API Gateway
- Find AWS Access Key
- Loot other new student data

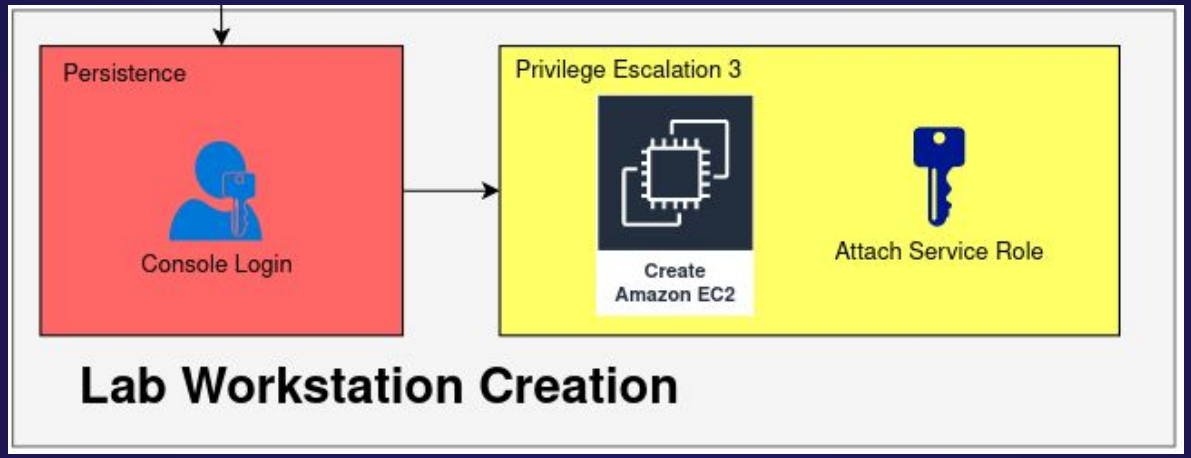


Attack Path 1



- Student Advisor Workstation Creation
  - Access Systems Manager with AWS Access Key
  - Find SSM command in the Systems Manager with hardcoded credentials
- After Gaining Access to an EC2
  - Login to RDS Database
  - Loot Student's personal information, financial info, etc.

Attack Path 1



- Lab Workstation Creation
  - Access AWS Console with hardcoded credentials
  - Misconfigured Role policies
    - Can attach a higher role to a EC2 Instance
  - Create an EC2 Instance and attach a role that grant complete AWS control

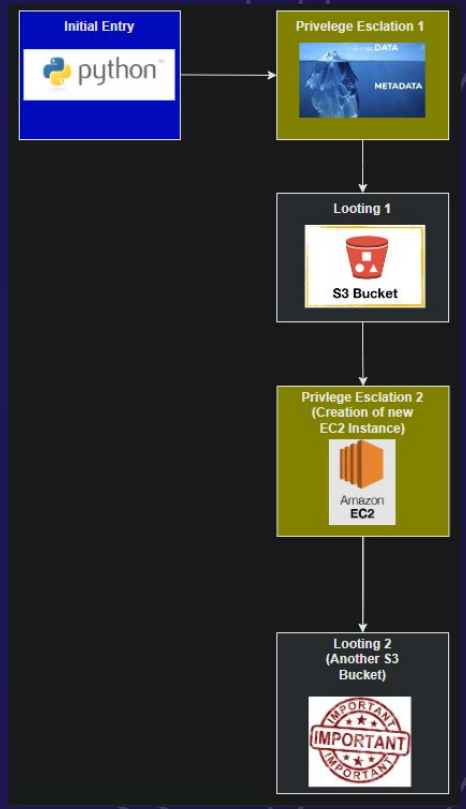
Final Design - Attack Path 2

Based on Red Team Methodology

5 sections follow methodology
- Initial Entry SSRF
- Privilege Escalation 1
- Looting 1
- Privilege Escalation 2
- Looting 2

Components leverage IMDS
- Widely used metadata service in AWS
- Based on misconfigured services

Certain components based on real life attacks
- Initial Entry based on Capital one attack
- Priv Esc 2 based on United hack



System Design - Attack Path 2

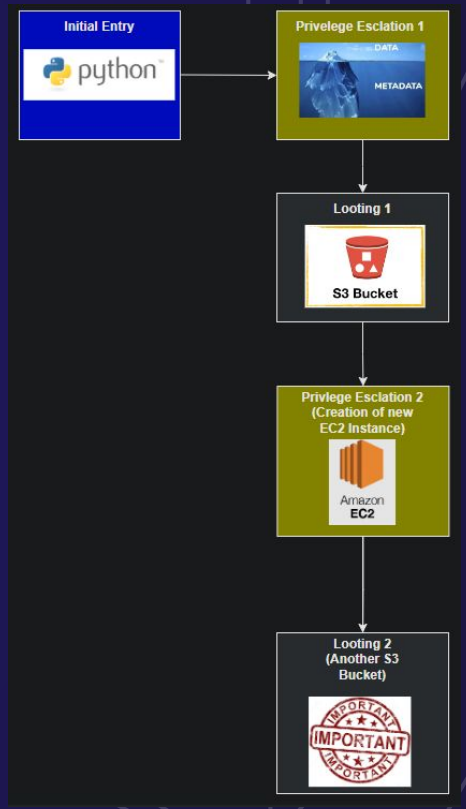
Initial Entry SSRF
- Utilizing an SSRF attack on an Django web server

Privilege Escalation 1
- Metadata api
- Temp credentials used to rollback policies

Looting 1
- Loot credentials from S3
- Using passrole permissions

Privilege Escalation 2
- Create new Ec2 instance, passrole more privileged role to ec2

Looting 2
- Loot sensitive information from second S3 with new privileged role



Implementation & Testing Process

- Proof of concept - Design
- Expanded on proof of concept - implementation
- Collaborated to connect component groups into complete Attack Path
- Tested individual stacks for bug fixes and improvements
- CloudFormation Templates after passing unit tests
- Conducted full system testing for deployment, functionality, and deletion
- Simplified non-AWS resource upload: Retrieved files from GitHub and uploaded them through manual lambda function triggers after initial stack deployment

AP1 Deploy to Priv-Esc1 Demo

Deploying the Stack

```
"Resources": {
    "PostDataRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "RoleName": "PostDataRole",
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "lambda.amazonaws.com"
                        },
                        "Action": "sts:AssumeRole"
                    },
                    {
                        "Sid": "Statement1",
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "iam.amazonaws.com"
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "PermissionsBoundary": "arn:aws:iam::308130987840:policy/Boundaries",
            "Path": "/",
            "Policies": [
                {
```

Resources (45)

Search resources

Logical ID	Physical ID	Type	Status
FillRDS	AP1-FullStack-FillRDS-XNUFFzmueGxk	AWS::Lambda::Function	CREATE_COMPLETE
InstanceSecurityGroup	sg-0bc8cfaaf65e3360b	AWS::EC2::SecurityGroup	CREATE_COMPLETE
InternetGateway	igw-01bd77061fe4fd728	AWS::EC2::InternetGateway	CREATE_COMPLETE
InternetGatewayAttachment	IGW|vpc-04cb35935b2d4055d	AWS::EC2::VPCGatewayAttachment	CREATE_COMPLETE
LambdaExecutionRole	AP1-FullStack-LambdaExecutionRole-38zmZCMdjhzT	AWS::IAM::Role	CREATE_COMPLETE
NatGateway1	nat-059d2e9fe82927647	AWS::EC2::NatGateway	CREATE_COMPLETE
NatGateway1EIP	18.219.233.141	AWS::EC2::EIP	CREATE_COMPLETE
persistenceUser	bnoel	AWS::IAM::User	CREATE_COMPLETE
PopLambda	PopulateS3Lambda	AWS::Lambda::Function	CREATE_COMPLETE
PopulateLambdaRole	PopulateLambdaRole	AWS::IAM::Role	CREATE_COMPLETE
PostDataRole	PostDataRole	AWS::IAM::Role	CREATE_COMPLETE

CloudFox Enumeration

Validate user profile creation:

```
ashlerbenda@pop-os:~$ aws sts get-caller-identity --profile ApiUser
{
    "UserId": "[redacted]",
    "Account": "[redacted]",
    "Arn": "arn:aws:iam::[redacted]:user/ApiUser"
}
```

Command to get ALL information immediately available to the user about the environment

```
~$ cloudfox aws --profile ApiUser all-checks
```

Available Resources

We find that our initial user has access to some API endpoints

```
ashlerbenda@pop-os:~/.cloudfox/cloudfox-output/aws/ApiUser-[redacted]/loot$ ls
api-gw.txt  endpoints-UrlsOnly.txt  inventory.txt  pull-secrets-commands.txt
ashlerbenda@pop-os:~/.cloudfox/cloudfox-output/aws/ApiUser-[redacted]/loot$ cat endpoints-UrlsOnly.txt
https://[redacted].execute-api.us-east-2.amazonaws.com/poc/getobj
https://[redacted].execute-api.us-east-2.amazonaws.com/prod/students
```

We can even see what syntax the endpoints expects the body to follow

```
ashlerbenda@pop-os:~/.cloudfox/cloudfox-output/aws/ApiUser-[redacted]/loot$ cat api-gw.txt
curl -X POST https://[redacted].execute-api.us-east-2.amazonaws.com/poc/getobj -H 'Content-Type: application/json' -d '{}'
curl -X GET https://[redacted].execute-api.us-east-2.amazonaws.com/prod/students
curl -X POST https://[redacted].execute-api.us-east-2.amazonaws.com/prod/students -H 'Content-Type: application/json' -d '{}'
```

Invoking the API Gateway

POST Data into S3 Bucket to retrieve hint

```
┌──(kali㉿kali)-[~]
└─$ aws apigateway test-invoke-method --profile ApiUser --region us-east-2 --rest-api-id [redacted] --resource-id [redacted] --http-method POST --path-with-query-string "/students" --body '{"name": "John Do
23-456-7890", "email": "john.doe@example.com", "ssn": "123-45-6789"}'
{
    "status": 200,
    "body": "\"Thank you for registering to the Virtual University of Ludicrous Notions (VULN)! Please use the GET method with student-9.json as the object value to retrieve student data as needed.\"",
    "headers": {
        "X-Amzn-Trace-Id": "Root=1-662adc28-5f579c9634f79ee787d83ea2;Parent=10b8af1cb7b7b311;Sampled=0;lineage=0680a311:0"
    },
    "multiValueHeaders": {
        "X-Amzn-Trace-Id": [
            "Root=1-662adc28-5f579c9634f79ee787d83ea2;Parent=10b8af1cb7b7b311;Sampled=0;lineage=0680a311:0"
        ]
    }
```

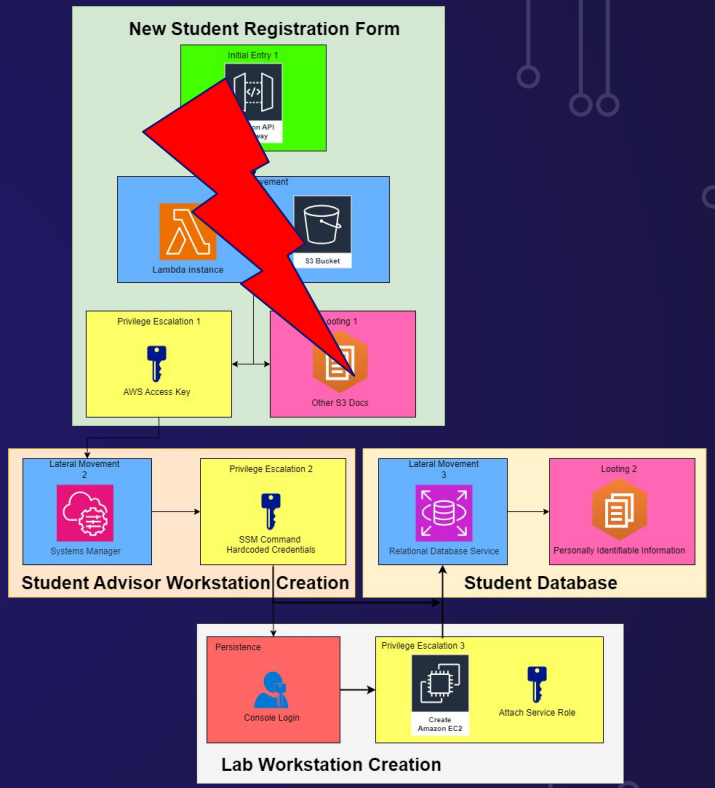
GET Unauthorized data from S3 Bucket

```
┌──(kali㉿kali)-[~/.aws]
└─$ aws apigateway test-invoke-method --profile ApiUser --region us-east-2 --rest-api-id [redacted] --resource-id [redacted] --http-method GET --path-with-query-string "/students?object_key=student-4.json"
    "status": 200,
    "body": "{\"name\": \"Emily Brown\", \"address\": {\"street\": \"101 Pine St\", \"city\": \"Dallas\", \"state\": \"TX\", \"zipcode\": \"54321\"}, \"phone_number\": \"765-432-1098\", \"email\": \"emi
    "headers": {
        "X-Amzn-Trace-Id": "Root=1-662ac74c-630b7c1812cb46da36d29b91;Parent=035c491f19eacb8a;Sampled=0;lineage=0680a311:0"
    },
    "multiValueHeaders": {
        "X-Amzn-Trace-Id": [
            "Root=1-662ac74c-630b7c1812cb46da36d29b91;Parent=035c491f19eacb8a;Sampled=0;lineage=0680a311:0"
        ]
    },
```

And so forth…

- User continues the attack path until full account compromise
  - Cloudfox will continually be used to help identify vulnerabilities and misconfigurations

- After full account compromise, the stack can be deleted as a whole in two simple steps



Conclusion

- Deliverables to client
  - Two attack paths created from Cloudformation Templates
  - Walkthrough guides and setup instructions

- End deliverables are focused on ease of use. Documentation are verbose including usage and technical details.

- Our stacks are extremely lightweight allowing for quick deployment and easy teardown.

Questions?