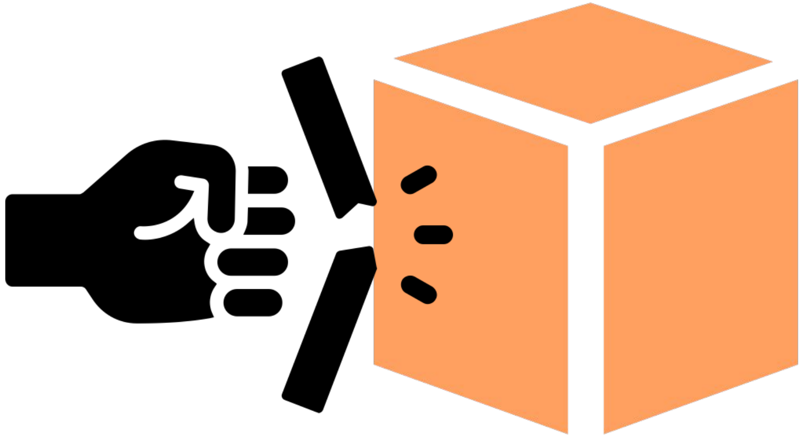


Pentesting AWS 101

Get Ready for Hands-on AWS Security

Who IAM

Sanjeev Jaiswal (Jassi)

12+ years of Experience in

- Application Security
  - Threat Modeling
  - Secure Code Review
  - Secure Code Training
- Cloud Security
- Security Automation using Python3
- Pentest & OWASP Top 10

What you know in AWS

Minimal AWS assumptions

- Can access AWS Console access
- Good idea of IAM features
- EC2 related operations
- How to work with S3 buckets
- Aware of ELB
- Security group, NACL
- API gateway
- Understanding of Lambda
- Various topics in VPC
- Security of|in the cloud

What I will cover

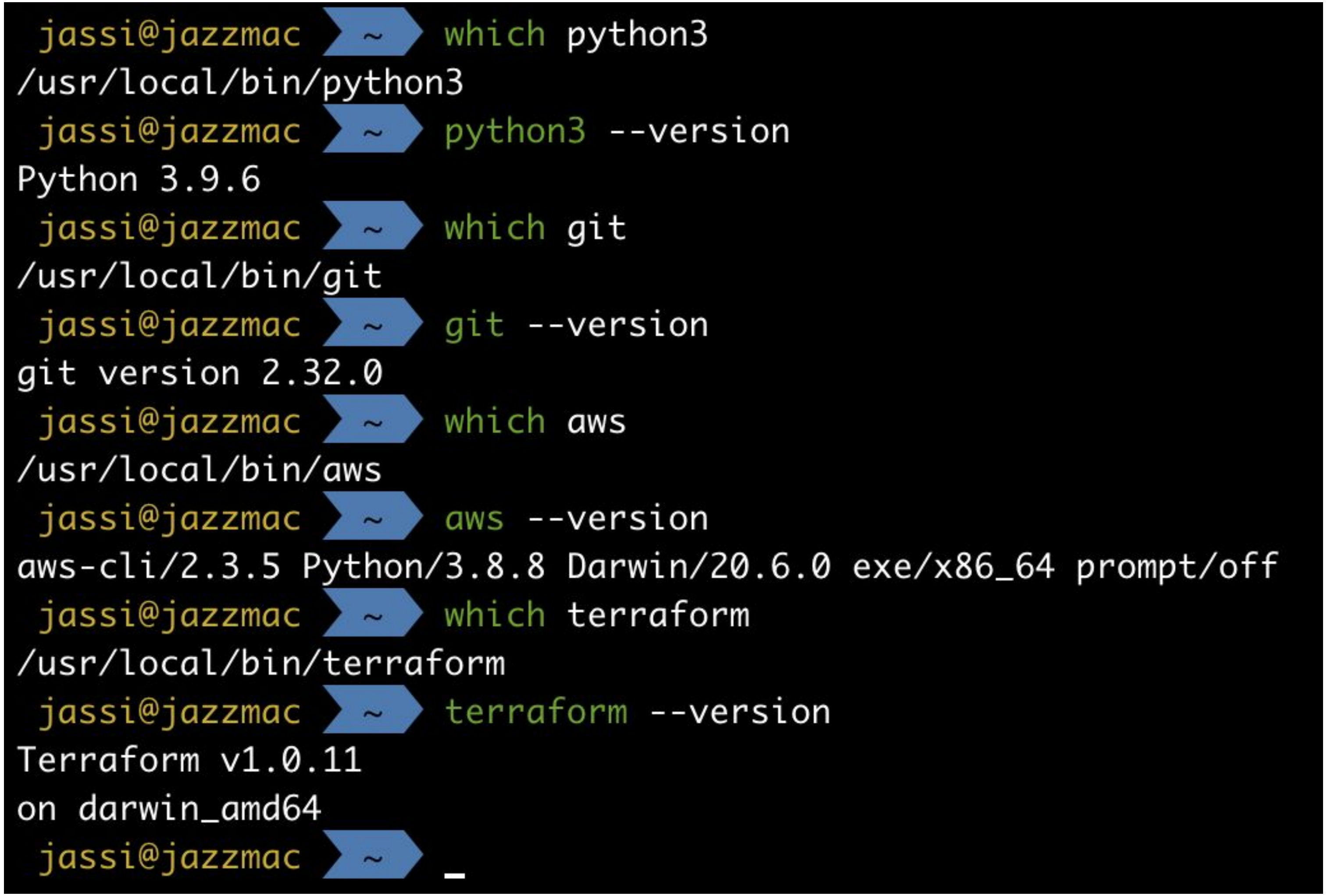
Fundamentals

- IAM pitfalls
- S3 Security
- IMDSv1 vs IMDSv2
- Vulnerable Public IPs, ELBs, Endpoints
- Lambda Security (Intermediate)
- ECS Security (Advanced)

Hands-On

- Flaws.cloud (1&~~2~~)
- CloudGoat Scenarios
- Pacu
- Prowler

Prerequisites: Python3, Terraform, aws-cli, git



```
 jassi@jazzmac  ~  which python3
/usr/local/bin/python3
 jassi@jazzmac  ~  python3 --version
Python 3.9.6
 jassi@jazzmac  ~  which git
/usr/local/bin/git
 jassi@jazzmac  ~  git --version
git version 2.32.0
 jassi@jazzmac  ~  which aws
/usr/local/bin/aws
 jassi@jazzmac  ~  aws --version
aws-cli/2.3.5 Python/3.8.8 Darwin/20.6.0 exe/x86_64 prompt/off
 jassi@jazzmac  ~  which terraform
/usr/local/bin/terraform
 jassi@jazzmac  ~  terraform --version
Terraform v1.0.11
on darwin_amd64
 jassi@jazzmac  ~  _
```

Installed & Verified

Pentesting AWS

How it's different from traditional Pentest?

- Infrastructure
  - External
  - Internal
- Environment
  - Hybrid
  - Native
- Application Implementation
  - Serverless/Lambda
  - Beanstalk
  - Lightsail …
- Data Storage (S3, RDS …)
- Security Misconfigurations (IAM)

Rules for Pentest in AWS

- Testing AWS Infrastructure
- No DoS and DDoS without notification
- No Port flooding
- No Protocol flooding
- No Network Stress testing without Notification
- Mail if you are unsure:
  - aws-security-simulated-event@amazon.com

Pacu/CloudGoat Testing environments follow AWS security Best Practices

Testing Customer Owned Application

Free to use any Pentest Tools

8 permitted Amazon Services

- Amazon EC2 instances, NAT Gateways, and Elastic Load Balancers
- Amazon RDS
- Amazon CloudFront
- Amazon Aurora
- Amazon API Gateways
- AWS Lambda and Lambda Edge functions
- Amazon Lightsail resources
- Amazon Elastic Beanstalk environments

https://aws.amazon.com/security/penetration-testing/

IAM Security issues

- Permissive policies
  - Resource policy: No * please
  - IAM policy: put*, attach*, policy versioning, group|user management related policies
  - Service Role Trust Policy
- Cross-Account Permissions
- Misconfigured IAM Role combination
- Poorly written policies
- Published without proper validation and review

S3 Security issues

- Publicly listed buckets and objects majorly in us-east-1 and eu-west-1 region
- Authorised to any AWS user
- Authorised to any AWS user in a specific region
- Common bucket enumeration
- No Cloudfront or WAF to avoid direct bucket exposure
- Default Encryption is not enabled
- No https enforcement for put objects
- Permissive bucket policy or bucket policy missing
- Virtual hosted: `https://bucketname.s3.Region.amazonaws`
- Path Style: `https://s3.Region.amazonaws.com/bucket-name/`

IMDSv1 vs IMDSv2

- IMDS: Instance Metadata Service
- Instance Metadata Service Version 1 (IMDSv1) – a request/response method
- Instance Metadata Service Version 2 (IMDSv2) – a session-oriented method
- By default you can use either IMDSv1 or IMDSv2 or both
- IMDS makes available metadata about the instance, its network, and its storage.
- It makes AWS creds available for any IAM role that is attached to the instance.
- SSRF attack is quite possible

Enforce IMDSv2: Policy based (Mitigation)

```
{
    "Version": "2012-10-17",
    "Statement": [
                {
            "Sid": "RequireImdsV2",
            "Effect": "Deny",
            "Action": "ec2:RunInstances",
            "Resource": "arn:aws:ec2:*:*:instance/*",
            "Condition": {
                "StringNotEquals": {
                    "ec2:MetadataHttpTokens": "required"
                }
            }
        }
    ]
}
```

Pentesting AWS - Demo

Exploit the misconfigurations

- Flaws.cloud
- Cloudgoat
  - Overview and tool walkthrough
  - iam_privsec_by_rollback
  - cloud_breach_s3
- Pacu
  - Introduction and setup
  - iam_privsec_by_rollback
  - ec2_ssrf

flaws.cloud

Get the real heat

- Url: http://flaws.cloud/
- It has total 6 levels
- Covering misconfigurations in
    - S3 (3 scenarios)
    - EC2 snapshot is made public
    - Reverse Proxy with IMDSv1 enabled
    - Lambda with excessive permission

Cloudgoat

Exploit the vulnerable settings

- Cloudgoat
  - By Rhino Security Labs
  - Vulnerable by design
  - Overview and tool walkthrough
- Demo of some scenarios
  - iam_privsec_by_rollback
  - cloud_breach_s3
  - ec2_ssrf
  - rce_webapp

How to use Cloudgoat

1. Once you clone the repo, go to cloudgoat directory
2. run `pip3 install -r ./core/python/requirements.txt`
3. Make sure cloudgoat.py is executable `chmod u+x cloudgoat.py`
4. run ./cloudgoat.py config profile and click y, then give profile name as default
5. whitelist your IP, run this command: `./cloudgoat.py config whitelist --auto` and type yes to continue
6. List all scenarios: ./cloudgoat.py list all|undeployed|deployed
7. Deploy the scenario: ./cloudgoat.py create scenario_name

Pacu - Cloud Pentesting Framework

Tool to make attackers job easier

- Pacu
  - Open-source AWS Exploitation Framework
  - By Rhino Security Labs
  - Written in Python
- Exploitation Demo
  - Introduction and setup
  - cloud_breach_s3
  - iam_privsec_by_rollback
  - ec2_ssrf
  - codebuild_secrets

How to use Pacu

1. You can use pacu for cloudgoat scenarios
    1. First deploy some scenarios to make pacu working easily
2. Once you clone the repo, go to pacu directory
3. Run cli.py as python3 cli.py or ./cli.py
4. Set a new session name or use existing one
5. Import all keys or import only one key `import_keys --all`
6. choose which key you want to import
    1. import_keys <profile-name>
    2. run whoami
7. Check if pacu settings are working. Run aws sts get-caller-identity --profile <profile-name>
8. See what modules are available in pacu: `list` (run once import key step is done)

What’s Next

- AWS Security Automation: https://github.com/awslabs/aws-security-automation
- SANS SEC573: https://www.sans.org/course/automating-information-security-with-python
- Automate event-driven security stuffs using AWS Lambda in Python
- Automate AWS Services security assessment using Python
- Automate AWS CIS benchmarks
- Automate AWS Exploits
- Automate/Solve AWS Based CTF challenges
- Use Pacu, Prowler, ScoutSuite for AWS Exploitation and Security Assessment
- Make command line tool using click module, similar to weirdAAL
- Breaking and Pwning Apps and Servers on AWS and Azure - Free Training Courseware and Labs

Resources

AWS Pentesting By PackT

AWS CLI by Amazon

Boto3 Documentation

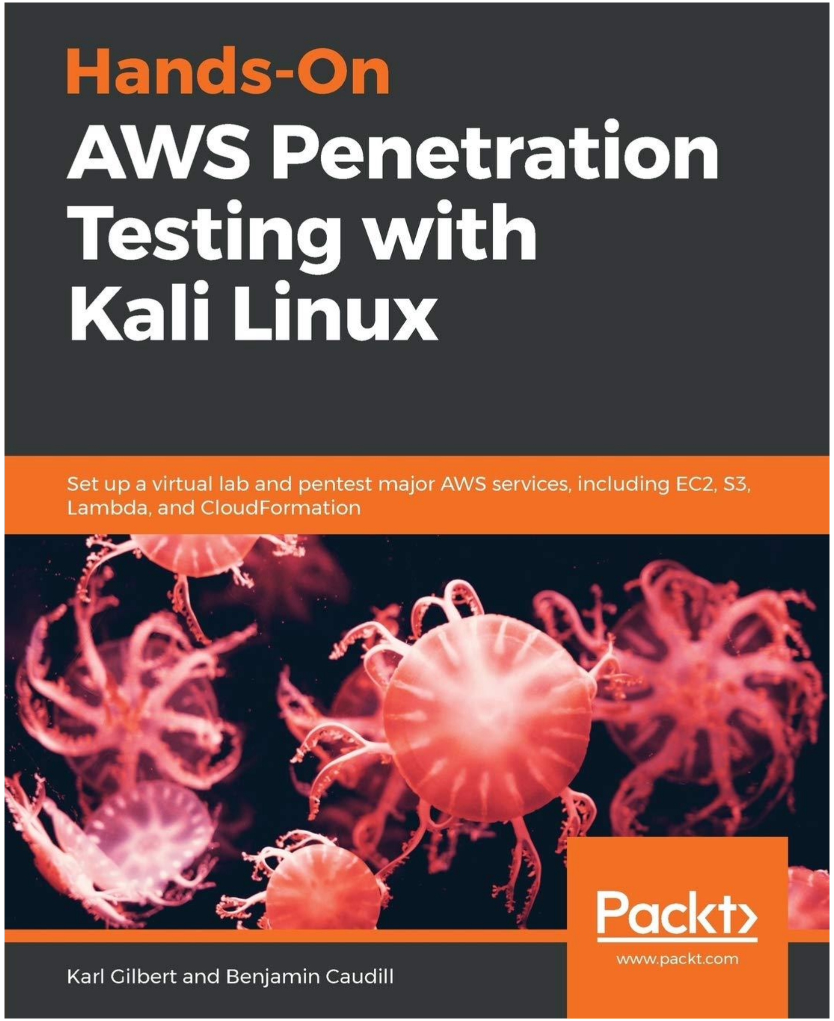
https://github.com/RhinoSecurityLabs/cloudgoat

https://github.com/RhinoSecurityLabs/pacu

https://github.com/toniblyx/my-arsenal-of-aws-security-tools

https://sra.io/blog/aws-iam-exploitation/

https://github.com/jassics/awesome-aws-security



Hands-On AWS Penetration Testing with Kali Linux

Set up a virtual lab and pentest major AWS services, including EC2, S3, Lambda, and CloudFormation

Packt>

www.packt.com

Karl Gilbert and Benjamin Caudill

My Social Handles

@jassics

/in/jassics

jassics@gmail.com

github.com/jassics

youtube.com/c/Flexmind

Credits

Image Credit

- Noun Project
- Flaticon

Content Credits

- Aws-well-architected-labs
- Boto3 Doc
- Cloudgoat and pacu