



# Mobile Application Security Assessments Automation

In today's digital landscape, ensuring the security of mobile applications is paramount. As mobile app usage explodes, so do the threats against them. This presentation explores the powerful combination of MobSF (Mobile Security Framework) and Jenkins pipelines to automate mobile app security assessments, providing a robust defense against potential vulnerabilities.

**SK** by Sheshananda Reddy Kandula

# Whoami

@Sheshananda Reddy Kandula

15 years in Application Security



**Sheshananda Reddy Kandula**

Sr Security Engineer at Adobe | AppSec |  
Product Security | OSWE | OSCP | CISSP



# Agenda

- Introduction to Mobile Application Security
- Mobile OWASP Top 10 Risks
- SAST and DAST
- MobSF
  - Installation
  - Docker
  - API Usage
- Jenkins
  - Jenkins features
  - Triggering the scans
- Conclusion

# Introduction to Mobile Application Security

- Mobile apps are vulnerable to security threats (e.g., malware, insecure storage, etc.).
- Regular assessments are critical but can be time-consuming.
- Automation bridges the gap between development speed and security compliance.

# Mobile OWASP Top 10 Risks



<https://owasp.org/www-project-mobile-top-10/>

# SAST vs DAST

What is SAST?

Definition: Static Application Security Testing (SAST) analyzes the source code, binary, or bytecode of mobile applications (Android/iOS) to identify vulnerabilities early in development without executing the application.

Methodology:

## 1. Code Review:

Scan source code for insecure coding practices (e.g., hardcoded credentials, insecure cryptography).

## 2. Configuration Analysis:

Analyze mobile app configurations (e.g., AndroidManifest.xml, Info.plist).

## 3. Data Flow Analysis:

Trace sensitive data (e.g., PII) to ensure it's stored and transmitted securely.

## 4. Dependency Scanning:

Identify vulnerabilities in third-party libraries and SDKs.

## 5. Security Rules and Policies:

Compare code against security guidelines (e.g., OWASP Mobile Top 10).

# MobSF

- Installation
- Docker
- API Usage

## [Link](#)

Installation:

```
docker pull opensecurity/mobile-security-framework-mobsf:latest
```

```
docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest
```

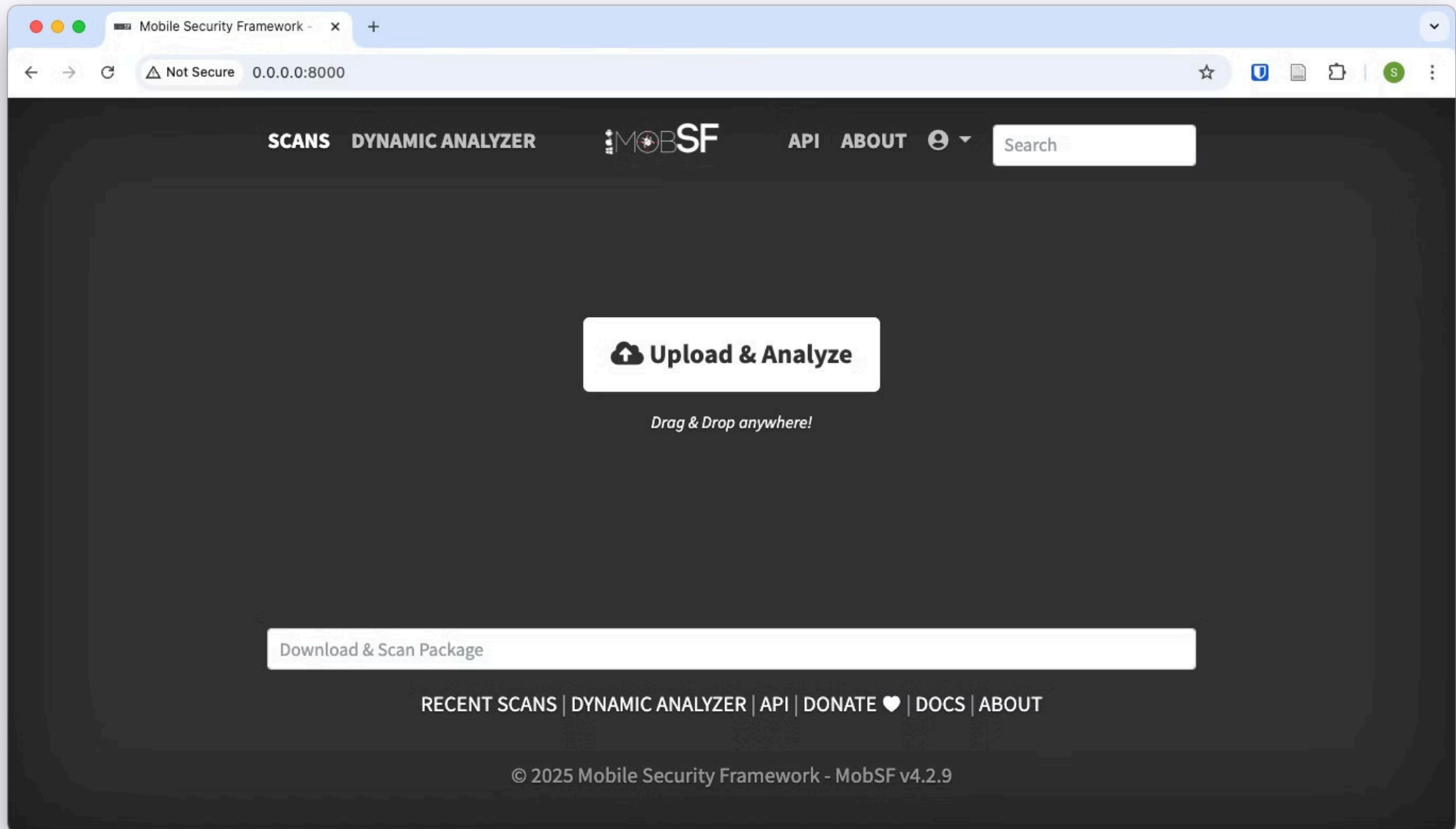
```
[2025-01-14 00:32:28 +0000] [1] [INFO] Starting gunicorn 23.0.0
[2025-01-14 00:32:28 +0000] [1] [INFO] Listening at: http://0.0.0.0:8000 (1)
[2025-01-14 00:32:28 +0000] [1] [INFO] Using worker: gthread
[2025-01-14 00:32:28 +0000] [118] [INFO] Booting worker with pid: 118
[INFO] 14/Jan/2025 00:32:28 - Loading User config from: /home/mobsf/.MobSF/config.py
[INFO] 14/Jan/2025 00:32:52 -
```

```
-- -- - - - - -
| V | | | | | | | | | | | | | | |
| M | | | | | | | | | | | | | | |
| | | | ( ) | | | | | | | | | | |
| | | | | | | | | | | | | | | |
```

```
[INFO] 14/Jan/2025 00:32:52 - Author: Ajin Abraham | opensecurity.in
```

```
[INFO] 14/Jan/2025 00:32:52 - Mobile Security Framework v4.2.9
```

# MobSF





# The Power of MobSF

## Static Analysis

MobSF delves into the application's source code, configurations, and permissions, uncovering vulnerabilities like insecure storage, hardcoded secrets, or excessive permissions.

## Dynamic Analysis

MobSF scrutinizes the application's runtime behavior, identifying risks like insecure API calls, unencrypted traffic, and data leakage, providing a complete picture of the app's security posture.

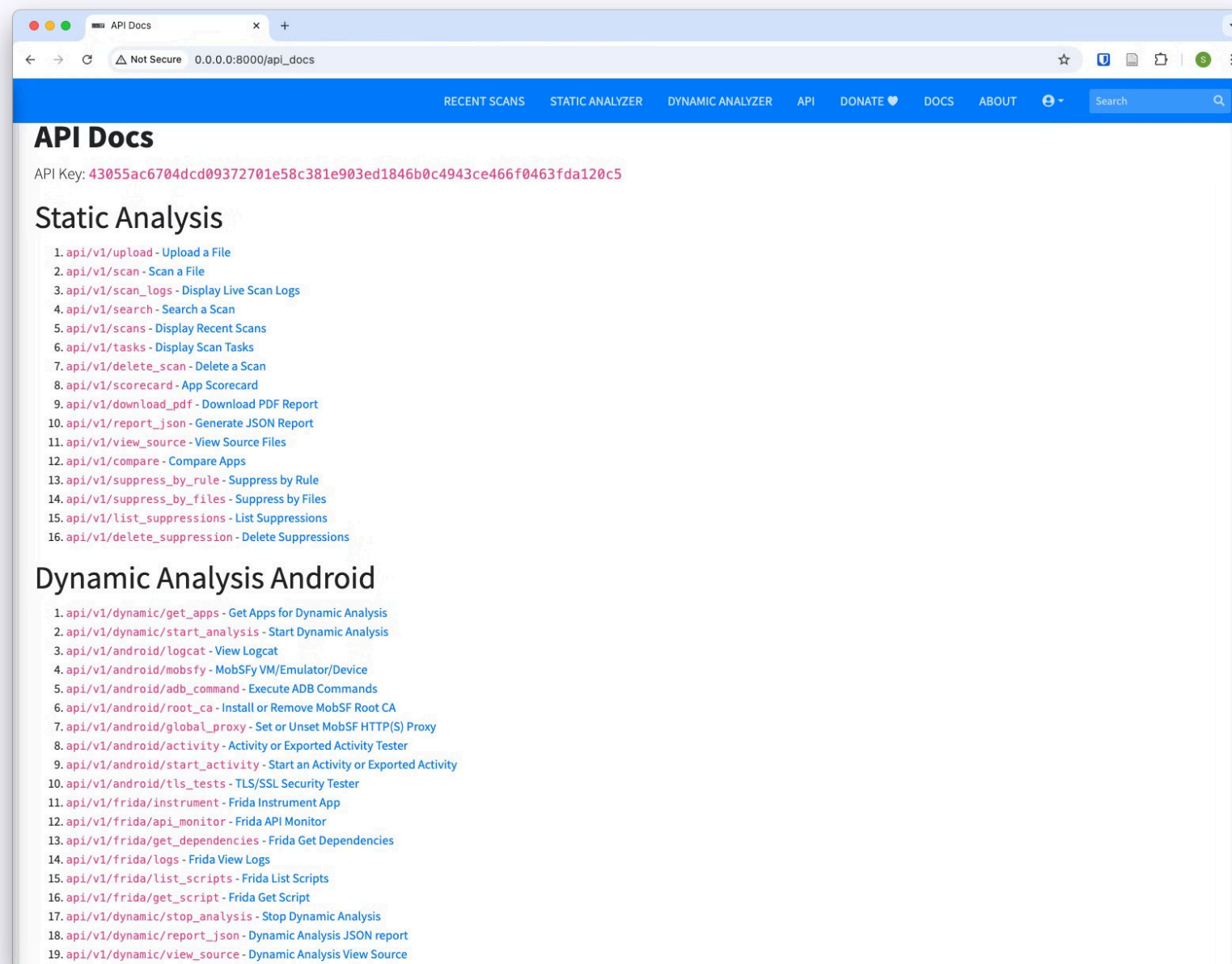
# MobSF: Your Mobile Security Swiss Army Knife

## API Support

MobSF's API enables seamless integration with automation tools like Jenkins, empowering you to trigger scans, retrieve reports, and manage the entire security assessment process programmatically.

## Comprehensive Reporting

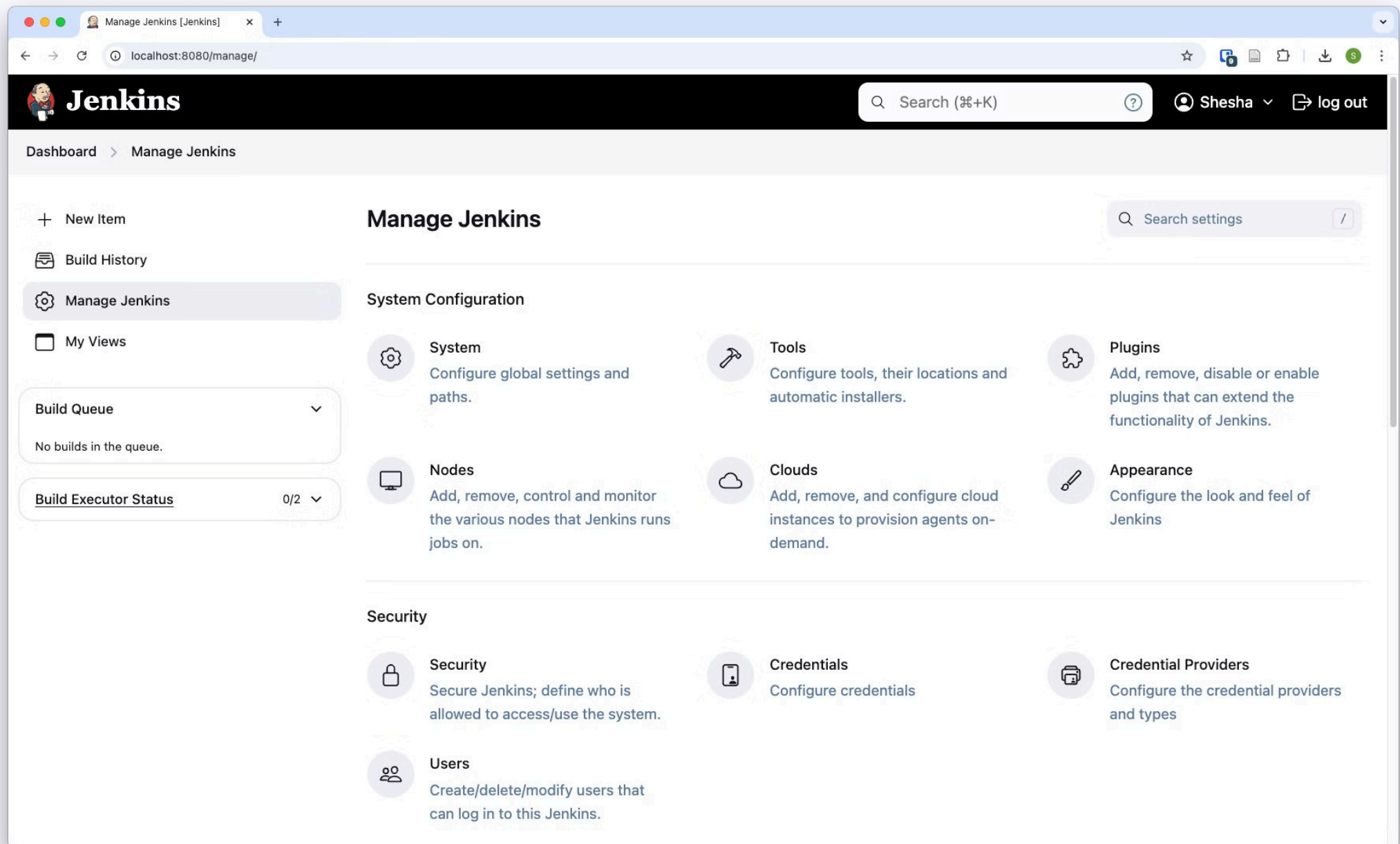
MobSF generates detailed, insightful reports, providing actionable information on identified vulnerabilities, their severity, and potential mitigation strategies.



# Jenkins

Download War file [WAR file](#)

run the following command: `java -jar jenkins.war --enable-future-java`



The screenshot shows the Jenkins web interface in a browser window. The browser's address bar displays 'localhost:8080/manage/'. The page header includes the Jenkins logo, a search bar with the text 'Search (⌘+K)', and a user profile for 'Shesha' with a 'log out' button. The main content area is titled 'Manage Jenkins' and features a search bar for settings. On the left sidebar, there are navigation options: '+ New Item', 'Build History', 'Manage Jenkins' (highlighted), and 'My Views'. Below these are two status boxes: 'Build Queue' showing 'No builds in the queue.' and 'Build Executor Status' showing '0/2'. The main content is organized into two sections: 'System Configuration' and 'Security'. The 'System Configuration' section includes: 'System' (Configure global settings and paths), 'Tools' (Configure tools, their locations and automatic installers), 'Plugins' (Add, remove, disable or enable plugins that can extend the functionality of Jenkins), 'Nodes' (Add, remove, control and monitor the various nodes that Jenkins runs jobs on), 'Clouds' (Add, remove, and configure cloud instances to provision agents on-demand), and 'Appearance' (Configure the look and feel of Jenkins). The 'Security' section includes: 'Security' (Secure Jenkins; define who is allowed to access/use the system.), 'Credentials' (Configure credentials), 'Credential Providers' (Configure the credential providers and types), and 'Users' (Create/delete/modify users that can log in to this Jenkins).



# The Need for Automation

## 1 Efficiency Gains

Automating security assessments streamlines the process, freeing up valuable developer time for addressing critical security issues.

## 2 Consistency and Scalability

Automated testing ensures consistent security evaluation across all builds, regardless of developer or project, guaranteeing a high standard of security across your mobile app portfolio.

## 3 Early Detection and Mitigation

Identifying vulnerabilities early in the development cycle allows for timely remediation, significantly reducing costs and minimizing the risk of vulnerabilities reaching production.

# Integrating MobSF with Jenkins Pipelines

## Pipeline Configuration

Configure Jenkins to automatically trigger MobSF scans whenever a new build (APK/IPA) is generated, ensuring continuous security evaluation throughout the SDLC.

## Threshold Policies

Jenkins monitors MobSF scan results and implements pre-defined security thresholds, halting builds that contain critical vulnerabilities and preventing the release of insecure apps.

1

2

3

4

## Security Scan Execution

MobSF performs both static and dynamic analysis on the uploaded build, conducting a comprehensive security assessment for each new release.

## Report Generation and Notifications

MobSF generates detailed reports, which are processed by Jenkins and automatically disseminated to stakeholders via email or issue trackers like Jira, enabling rapid feedback and issue resolution.

# Jenkins Job

```
pipeline {
  agent any
  environment {
  }
  MobSFAPIKey = credentials('MobSFAPIKey') MobSFURL = 'http://localhost:8000/'
}
options {
  timestamps()
}
triggers{
  cron('@midnight')
}
stages {
  stage("Download the APK or IPA File:"){
    steps {
      script{
        echo "Downloading the APK or IPA File:"
        echo "MobSF URL:"+MobSFURL
        fileDwld="wget -q -O AndroGoat.apk http://localhost:8888/AndroGoat.apk"
        echo "${env.WORKSPACE}"
        sh "rm -f ${env.WORKSPACE}/AndroGoat.apk"
        sh(script: fileDwld, returnStdout: false)
        echo("File Downloading completed")
      }
    }
  }

  stage("Starting MobSF Scan:"){
    steps {
      script{
        echo "Initiating the MobSF Scan:"
        // scan = "curl -X POST --url http://localhost:8000/api/v1/tasks -H \"Authorization:
d27147caa4ee637d022e501d7b475cf3d55e0cf57ca05eeee55e1f4e37fc872a\""

        //Display Recent Scans API

        uploadFile = "curl -F 'file=@${env.WORKSPACE}/AndroGoat.apk' \"http://localhost:8000/api/v1/upload\" -H
\"Authorization: d27147caa4ee637d022e501d7b475cf3d55e0cf57ca05eeee55e1f4e37fc872a\" -o response.json"
        response=sh(script: uploadFile, returnStdout: true).trim()
        echo response

        hash = sh(script: "cat response.json | jq -r '.hash' ", returnStdout: true).trim()
        echo hash

        scan="curl -X POST --url http://localhost:8000/api/v1/scan --data \"hash="+hash+"\" -H \"Authorization:
d27147caa4ee637d022e501d7b475cf3d55e0cf57ca05eeee55e1f4e37fc872a\""
        echo "Scan command.."+scan
        response1=sh(script: scan, returnStdout: true)
        echo "Scan Started..."
        echo response1

        // scorecard="curl -X POST --url http://localhost:8000/api/v1/scorecard --data \"hash="+hash+"\" -H
\"Authorization: d27147caa4ee637d022e501d7b475cf3d55e0cf57ca05eeee55e1f4e37fc872a\" -o response.json "
        scorecard="curl -X POST --url http://localhost:8000/api/v1/scorecard --data \"hash="+hash+"\" -H
\"Authorization: d27147caa4ee637d022e501d7b475cf3d55e0cf57ca05eeee55e1f4e37fc872a\" "
        echo "Scorecard: "+scorecard
        response2=sh(script: scorecard, returnStdout: true)
        echo "Scorecard ..." +response2

        report="curl -X POST --url http://localhost:8000/api/v1/download_pdf --data \"hash="+hash+"\" -H
\"Authorization: d27147caa4ee637d022e501d7b475cf3d55e0cf57ca05eeee55e1f4e37fc872a\" -o report.pdf"
        echo "report:"+report
        resposne3=sh(script: report, returnStdout: true)
        echo "Report is downloaded"
      }
    }
  }
}
}
```

# Benefits of Automating Mobile Security Assessments

## Efficiency and Focus

Automation significantly reduces manual effort by up to 60%, allowing security teams and developers to focus on high-priority issues and strategic initiatives.

## Scalability and Consistency

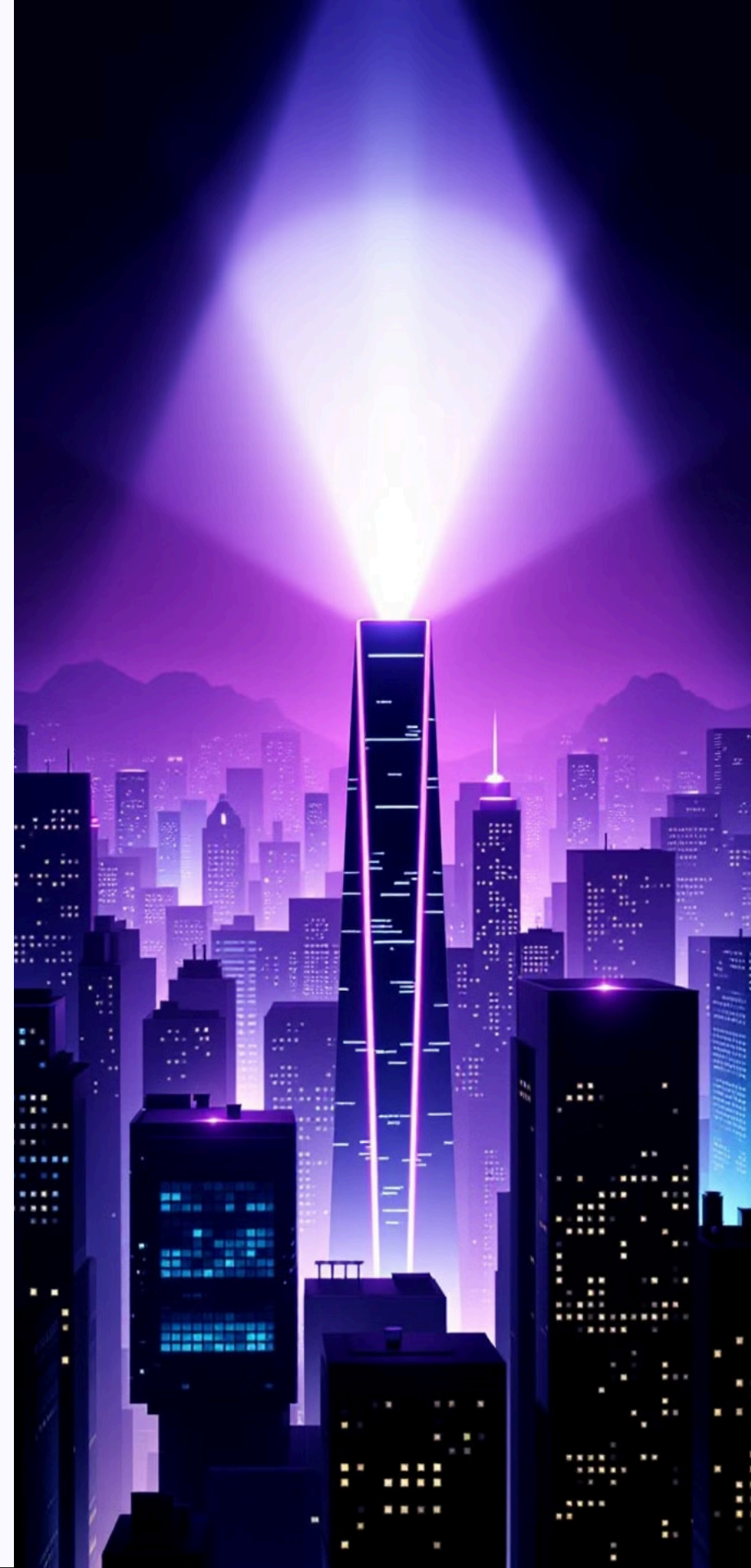
Automate security testing for a multitude of mobile applications concurrently, ensuring consistent security evaluation across your entire portfolio.

## Improved Security Posture

Early identification and remediation of vulnerabilities during development strengthens your overall security posture, minimizing the risk of exploitation in production.

## Developer Empowerment

Provide developers with comprehensive, actionable insights through detailed reports, empowering them to address security flaws independently and contribute to a more secure development process.





# Key Takeaways



## Automate Security Assessments

Integrate MobSF with Jenkins to automate security testing, ensuring consistent, efficient evaluation for each build.



## Address Security Flaws Early

Catch vulnerabilities early in the development lifecycle, reducing costs and improving the security of your mobile applications.



## Embrace DevSecOps

Incorporate security as an integral part of your development practices, fostering a culture of shared responsibility and proactive security measures.



# MobSF: A Key Component of Your Mobile Security Arsenal

1 Static Analysis

---

2 Dynamic Analysis

---

3 API Integration

---

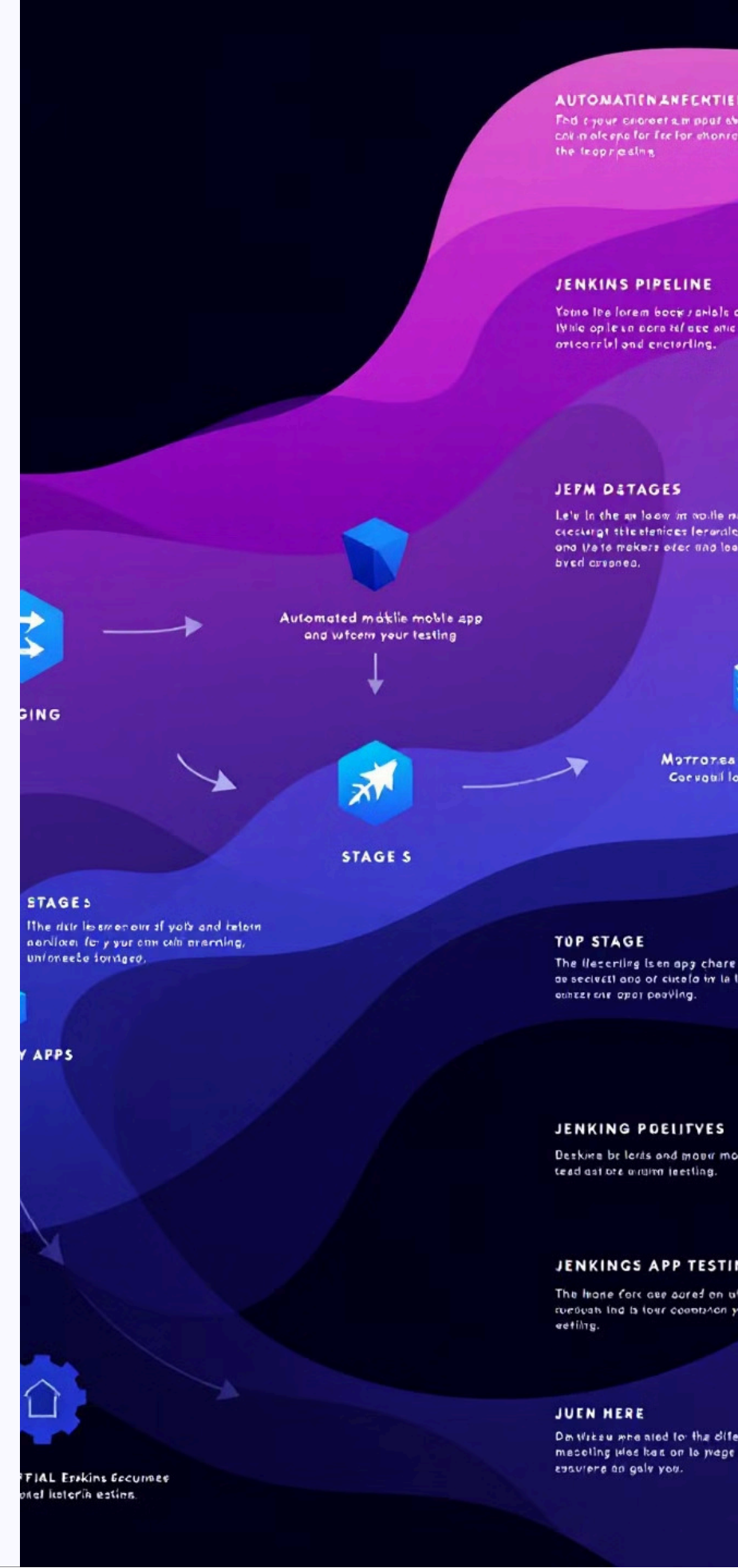
4 Comprehensive Reporting

MobSF offers a powerful suite of security assessment capabilities, enabling you to comprehensively evaluate the security of your mobile applications throughout the development lifecycle. Its API facilitates seamless integration with Jenkins, automating the assessment process for increased efficiency and scalability.

# Jenkins Pipelines: Driving Automation



Jenkins pipelines provide the framework for orchestrating the automated security assessment process, ensuring seamless integration with MobSF, continuous evaluation of builds, and timely notification of any vulnerabilities identified.



# Embracing Continuous Security Improvement



1

## Identify

Identify vulnerabilities through automated security assessments with MobSF and Jenkins.

2

## Remediate

Address vulnerabilities promptly, prioritizing critical issues and ensuring timely patching.

3

## Test

Retest the application after remediation to confirm the effectiveness of the fixes and ensure no new vulnerabilities have been introduced.

4

## Improve

Continuously improve security processes, incorporating feedback and lessons learned from previous assessments.



# Conclusion: Securing the Mobile Future

By leveraging the power of MobSF and Jenkins pipelines, organizations can effectively automate mobile application security assessments, ensuring robust, consistent security testing throughout the SDLC. This approach strengthens application security, empowers development teams, and supports the secure delivery of mobile applications in a rapidly evolving digital landscape. Embrace automation and DevSecOps principles to secure the mobile future.