Here's the full document in text format:

TIMS Security Assessment & Semgrep Evaluation Report

Prepared by: Pallavi (AppSec Intern) **Date:** June 2026 **Purpose:** Security posture assessment of the development team and evaluation of Semgrep as a SAST/SCA tool

Section 1: Current State of the Development Team

What the Team Has

The development team at TIMS has good developers who follow best coding practices and deploy applications on AWS. They have AWS Inspector running at the infrastructure level which catches some cloud configuration issues. Purple Team scans are conducted annually and bi-weekly security meetings take place between teams. The team is agile, moves fast, and delivers working software.

What the Team Lacks

1. Automated Security Scanning in the Code Pipeline Code is written and directly deployed to AWS with no automated tool checking it in between. There is no security gate that catches vulnerabilities before the code goes live. Developers are shipping code without any static analysis running on it.

2. CI/CD Pipeline There is no automated build and deployment process. Code is manually deployed to AWS which means there is no structured, repeatable, and secure process between writing code and pushing to production. Every deployment is a manual operation.

3. Shift Left Security Practice Security is not being brought in early enough in the development process. The developer himself admitted it is difficult to involve security early because requirements keep changing. This means security is often an afterthought rather than built in from the beginning. By the time security reviews happen, code is already written and fixing it is expensive.

4. SAST Tool — Static Application Security Testing Until Semgrep is introduced, there is no static analysis tool reviewing the code for security flaws, bad practices, or vulnerabilities while the code is being written or before it is deployed. No one is looking at the code from a security perspective before it goes live.

5. SCA — Software Composition Analysis There is no tool currently checking whether the third party libraries and dependencies used in the code have known vulnerabilities or CVEs. The team could be shipping code with vulnerable open source packages and have no visibility into it.

6. Consistent Security Requirements Early in Projects Because projects are agile and requirements change frequently, security requirements are never fully defined upfront. Security ends up being retrofitted into projects rather than designed in from the start. This creates technical debt and increases the cost of fixing issues later.

7. Formal Secure SDLC Process There is no structured Secure Software Development Life Cycle in place. Security checks, reviews, and gates are informal and inconsistent across projects rather than being a standard part of every project lifecycle.

Summary Table

Section 2: Current vs Future State

Current State

Developer writes code
↓
Manual AWS deployment
↓
No security checks anywhere
↓
Vulnerabilities go live 🚨

Future State with Semgrep

Developer writes code
↓
Semgrep runs locally (Shift Left) ← catches issues early
↓
Code committed to Bitbucket
↓
Semgrep runs in CI/CD pipeline ← automated security gate
↓
Only clean code gets deployed to AWS ✅
↓

Semgrep Cloud dashboard for visibility ← ongoing monitoring

Section 3: What We Did — Semgrep Evaluation

To evaluate Semgrep as a potential SAST and SCA tool for TIMS, I set up a complete security testing environment and ran scans on DVWA (Damn Vulnerable Web Application) — an intentionally vulnerable application used industry-wide for security testing and tool evaluation.

Environment Setup

- Installed UTM virtualization tool on Mac
- Created Windows 11 virtual machine inside UTM
- Enabled Hyper-V inside Windows 11
- Installed Miniconda (Python environment)
- Installed Bandit and Semgrep via pip
- Installed Git and cloned DVWA
- Forked DVWA to personal GitHub for cloud scanning

Section 4: Semgrep CLI Scan

Command Used

semgrep --config=auto C:\DVWA --json --output C:\dvwa_semgrep_report.json

What It Does

- --config=auto — automatically picks the best security rules
- --json — saves output as JSON for reporting
- --output — saves report to file

Results

- Files scanned: 250
- Rules applied: 289
- Findings: 78 vulnerabilities
- Languages covered: PHP, JavaScript, YAML
- Scan time: ~1 second

What CLI Shows

The CLI shows a progress bar, scan summary, and for each finding shows the rule name, file path, line number, vulnerability description, CWE reference, OWASP category, impact level, likelihood, and confidence level. Results are saved as a JSON report for further analysis.

Limitation of CLI

The CLI only uses free OSS rules, has no visual dashboard, cannot automatically detect false positives, and requires manual review of all findings.

Section 5: Semgrep Cloud Scan

How It Was Set Up

Forked DVWA to personal GitHub account, connected GitHub to Semgrep Cloud at semgrep.dev, and triggered a full scan from the Projects dashboard.

Results

- Files scanned: 250
- Total findings: 111
- Code findings: 108
- Supply chain findings: 3
- Scan time: 45 seconds

Priority Findings (Top 3 Critical)

Finding 1: taint-cookie-secure-false Severity: High Files: dvwa/dvwaPage.inc.php:226 and vulnerabilities/high.php:11 What it means: The secure flag on cookies is explicitly disabled. This causes cookies to be transmitted over unencrypted HTTP connections, allowing an attacker on the same network to intercept session tokens and hijack user sessions. Fix: Enable the secure and httponly flags on all cookies.

Finding 2: search-active-debug Severity: High File: vulnerabilities/impossible.php:6 What it means: Debug logging is explicitly enabled. In production this would expose sensitive information such as error messages, stack traces, database queries and file paths to potential attackers. Fix: Disable debug mode in all production environments. Use environment variables to control debug settings.

What Cloud Gives Extra

The cloud version automatically identified false positives, provided a visual dashboard with filters by severity and category, allowed triage of findings by marking them as Open, Ignored, or Fixed, and detected supply chain vulnerabilities that the CLI completely missed.

Section 6: CLI vs Cloud Comparison

Feature	Semgrep CLI	Semgrep Cloud
Total findings	78	111
Rules applied	289 OSS rules	Pro + OSS rules
Dashboard	No	Yes
False positive detection	Manual	Automatic
Triage capability	No	Yes
Supply chain scan	No	Yes
Setup required	Just CLI	GitHub connection
Scan time	~1 second	45 seconds
Cost	Free	Free tier available
Best for	Quick local scans	Team collaboration

Section 7: Vulnerability Categories Found

Vulnerability	Count	Severity	OWASP Category
SQL Injection	~15	High	A03 Injection
Command Injection	~12	Critical	A03 Injection
SSRF	~10	Medium	A10 SSRF
MD5 Loose Equality	~8	Low	A02 Cryptographic Failures
eval() Usage	~6	High	A03 Injection
Insecure Cookies	~2	High	A02 Cryptographic Failures
CORS Misconfiguration	~2	Low	A07 Auth Failures
phpinfo Exposure	~1	Medium	A01 Broken Access Control
Path Traversal	~1	Medium	A01 Broken Access Control
ReDoS	~2	Medium	A05 Misconfiguration
Debug Mode Enabled	~1	High	A05 Misconfiguration

GitHub Actions Injection	~1	High	A03 Injection

Section 8: False Positive Analysis

Out of 111 findings approximately 20-25 are likely false positives — roughly a 20% false positive rate. These are primarily found in impossible.php files which represent the already-secured version of DVWA code. Semgrep Cloud automatically flagged several of these with explanations, which saves significant triage time compared to the CLI where everything must be reviewed manually.

Section 9: What Semgrep Caught vs Missed

Caught by Semgrep ✅	Missed by Semgrep ❌
SQL Injection	CSRF
Command Injection	Business logic flaws
SSRF	Insecure file upload logic
XSS	Authentication bypass
CORS Misconfiguration	Session management issues
Insecure Cookies	Runtime vulnerabilities
Debug Mode Enabled	
Supply Chain vulnerabilities	

Semgrep is a static analysis tool — it can only find issues in the code itself. It cannot find vulnerabilities that only appear at runtime or logic-based flaws that require understanding of the business context. For complete coverage, Semgrep should be used alongside a DAST tool like OWASP ZAP.

Section 10: Semgrep Evaluation — Is It Right for TIMS?

Strengths

Semgrep directly addresses 5 out of 7 security gaps the development team currently has. It provides SAST coverage at the code level which AWS Inspector completely misses. It supports multiple languages including the Python and JavaScript stack used at TIMS. It integrates natively with GitHub and Bitbucket CI/CD pipelines. The cloud version provides a team dashboard, triage

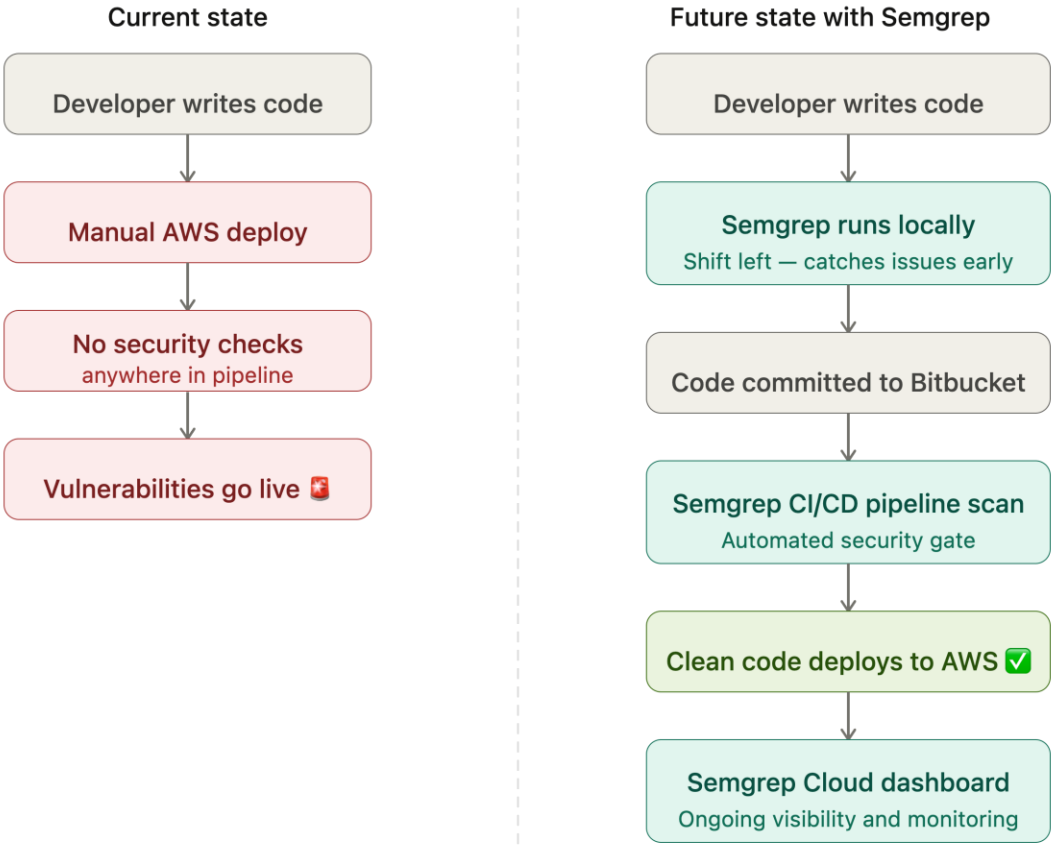
workflow, and supply chain scanning. It is easy to write custom rules targeting specific patterns relevant to TIMS code. It is fast — scanning 250 files in under a second locally.

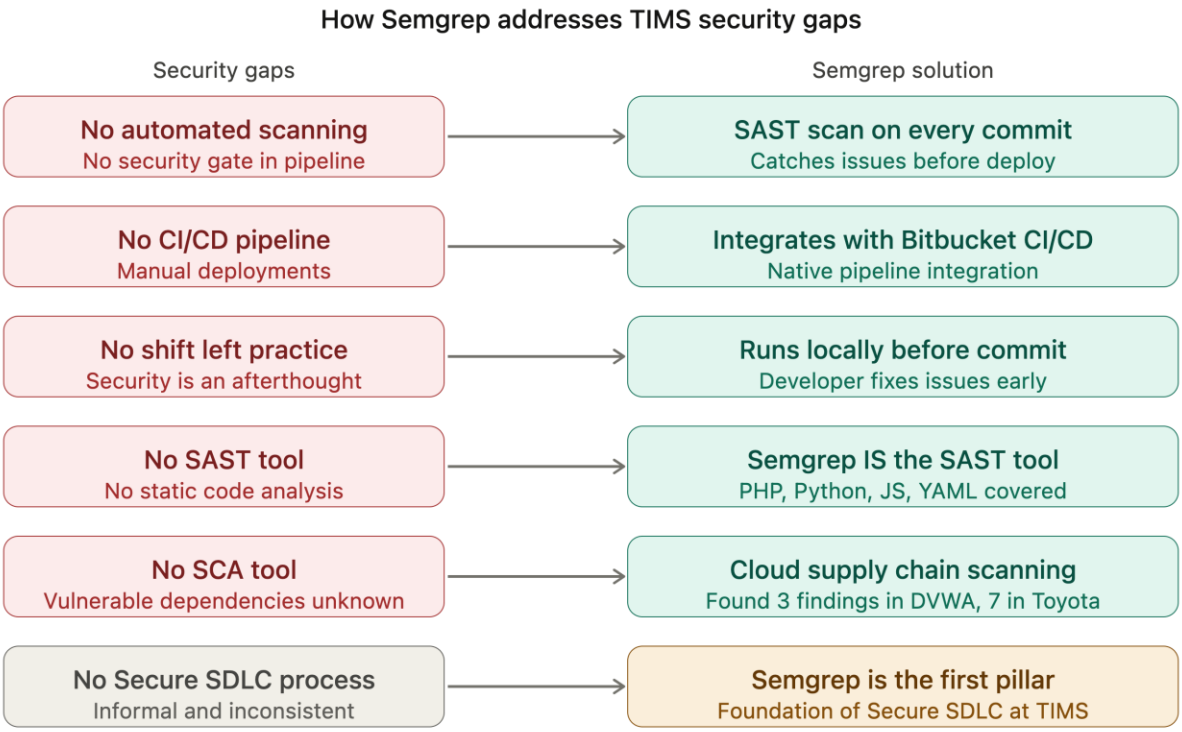
Limitations

Semgrep cannot detect runtime vulnerabilities or business logic flaws. The most powerful rules require a Pro subscription. Custom rules need to be written for TIMS-specific patterns like AWS credential handling. It needs to be paired with a DAST tool for full coverage. Java support requires Semgrep Pro for deep cross-file taint analysis which is relevant if TIMS has any Java services.

Recommendation

Yes — adopt Semgrep as the primary SAST and SCA tool for TIMS. It directly fills the biggest gaps in the current security posture and is the right foundation for building a Secure SDLC. The immediate next step is to integrate Semgrep into the Bitbucket CI/CD pipeline so every code commit is automatically scanned before it reaches AWS. This single change would address gaps 1, 2, 3, and 4 simultaneously and mark the beginning of a real DevSecOps practice at TIMS.


Current state
Developer writes code
Manual AWS deploy
No security checks
anywhere in pipeline
Vulnerabilities go live 🚨
Future state with Semgrep
Developer writes code
Semgrep runs locally
Shift left — catches issues early
Code committed to Bitbucket
Semgrep CI/CD pipeline scan
Automated security gate
Clean code deploys to AWS ✅
Semgrep Cloud dashboard
Ongoing visibility and monitoring



Here's the section to add to your document:

Section 12: Developer Adoption — How Easy Is Semgrep to Use?

For Developers — Ease of Use

One of the biggest concerns when introducing any security tool is whether developers will actually use it. A tool that is too complex or slows down development will be ignored or worked around. Semgrep scores very well here compared to most security tools.

Installing and Running Locally

Getting started with Semgrep takes less than 5 minutes. A developer installs it with a single command:

pip install semgrep

And runs a scan with:

semgrep --config=auto .

That is it. No complex configuration, no license server, no heavy IDE plugin required. Any developer comfortable with the command line can be up and running immediately.

Reading the Results

Semgrep output is written in plain English. Instead of cryptic error codes, it tells developers exactly what the problem is, which file and line it is on, and why it is dangerous. For example:

```
php.lang.security.injection.tainted-sql-string
File: login.php, Line 12
User data flows into a manually constructed SQL string.
This can allow an attacker to steal data from the database.
Fix: Use prepared statements instead.
```

A developer does not need security expertise to understand this. The message, the location, and the fix are all in the same output.

CI/CD Integration — One YAML Block

Adding Semgrep to a Bitbucket pipeline requires adding just this to the pipeline configuration:

```
- step:
    name: Semgrep Security Scan
    script:
      - pip install semgrep
      - semgrep --config=auto . --error
```

This is a one-time setup by DevOps. After this, every developer's code commit is automatically scanned without them needing to do anything differently.

What Changes for Developers Day to Day

This is the most important question for adoption. The honest answer is — very little changes for developers initially.

- They write code exactly as they do today
- When they push a commit, Semgrep runs automatically in the pipeline
- If there is a finding, they get a notification with the file, line, and fix recommendation
- They fix it and push again

The only friction is fixing the flagged issue before the code can merge. This is intentional — it is the security gate. But because Semgrep explains the issue clearly and suggests a fix, most developers find it educational rather than frustrating.

Potential Resistance Points

Honestly, there will be some resistance and it is important to acknowledge this upfront:

"It slows down our pipeline" Semgrep scans 250 files in under 1 second locally and 45 seconds in cloud mode. This is negligible compared to build and test times. This objection is unlikely to hold.

"Too many false positives" This is a valid concern. The CLI scan found approximately 20 false positives out of 111 findings — roughly 18%. Semgrep Cloud automatically identifies many of these. The team should spend time tuning rules and ignoring confirmed false positives in the first month so developers trust the output.

"We don't have time to fix security issues" This is a cultural challenge not a tool challenge. The answer is that catching issues now costs minutes. Catching them in production costs days. Prashant can address this at the management level.

"We don't understand the findings" Semgrep's plain English output helps significantly. Pairing it with short internal documentation explaining the top 5 most common findings at TIMS would remove most of this concern.

Recommendation for Smooth Adoption

To make adoption as smooth as possible, a phased rollout is recommended:

Phase 1 — Warn only (Month 1) Semgrep runs in the pipeline but does not block commits. Developers see the findings but can still merge. This builds awareness without creating friction.

Phase 2 — Block on critical only (Month 2) Only Critical and High severity findings block a merge. Medium and Low are warnings. Developers learn to address the most serious issues.

Phase 3 — Full enforcement (Month 3) All findings must be addressed or explicitly marked as false positives before merging. By this point developers are familiar with Semgrep and the process feels normal.

Bottom Line on Developer Adoption

Semgrep is one of the most developer-friendly security tools available. It is fast, clear, and integrates with minimal effort. The bigger challenge at TIMS will not be the tool itself — it will be the cultural shift of treating security as part of the development process rather than someone else's responsibility. That shift starts with a phased rollout, clear communication from leadership, and visible wins early on when Semgrep catches real issues before they reach production.

What a Full Secure CI/CD Proof of Concept Means

It means building a **complete automated security pipeline** where every time code is pushed, **multiple security checks run automatically** — not just Semgrep alone.

Developer pushes code to Bitbucket

↓

SAST Check → Semgrep scans code

for security vulnerabilities

↓

SCA Check → scans third party

libraries for known CVEs

↓

Secrets Check → looks for hardcoded

passwords, API keys in the code

↓

Everything passes?

↓

✅ Code is safe to deploy

❌ Issues found? Stop and alert developer

Let me pull up the last call transcript you shared.The last call transcript you shared in **this conversation** was the one with the **web/development team member** — here is the summary:

Last Call — Web/Development Team

Key Points Discussed:

1. Public Site Security — TFS Purple Team The person manages **toyoinsurance.com** and public facing sites. Security scanning is done by the **TFS Purple Team** — a Toyota Financial Services sister company that scans for exploits and vulnerabilities **every 6 months to a year.** Any issues found go into a tracker for remediation.

2. AWS Inspector Swetha and Naresh are involved in security audits using AWS Inspector for package vulnerability scanning. **Prashant leads** bi-weekly meetings to discuss findings and assign fixes, with urgent issues handled in daily standup.

3. Semgrep — Developer's Opinion The developer welcomed Semgrep but had not seen it yet. Their honest view was — having a static analysis tool means **one less thing to think about** for developers. Palavi reassured them it would be integrated slowly without disrupting their workflow.

4. Biggest Challenge — Security vs Agile The developer highlighted that bringing security in early is difficult because **requirements keep changing** throughout development. Security ends up being retrofitted rather than built in from the start — which is exactly the problem DevSecOps solves.