# Claude Code กับ DevSecOps ในมหาวิทยาลัย: แนวทางฝังการตรวจสอบความปลอดภัยในวงจรพัฒนาระบบ

*Claude Code and DevSecOps in Universities: Embedding Security Verification into the Development Lifecycle*

**พระมหาประสพสุข ปิยภาณี (สุขล้วน)**
นักวิชาการคอมพิวเตอร์ สำนักงานวิชาการ วิทยาลัยสงฆ์พุทธโสธร มหาวิทยาลัยมหาจุฬาลงกรณราชวิทยาลัย
Email: prasopsuk.suk@mcu.ac.th • DOI: 10.14457/MCU.IT.2569.007

**บทคัดย่อ (Abstract) —** การประยุกต์ใช้ตัวแทนปัญญาประดิษฐ์เพื่อการเขียนโค้ด (AI Coding Agents) เช่น Claude Code ในการพัฒนาระบบสารสนเทศระดับอุดมศึกษา ช่วยเพิ่มความเร็วในการผลิตซอฟต์แวร์ได้อย่างมีนัยสำคัญ แต่หากปราศจากการตรวจสอบความปลอดภัยที่รัดกุม ความเร็วนี้อาจกลายเป็นการเร่งสร้างช่องโหว่ซอฟต์แวร์ (Vulnerability Velocity) เข้าสู่สภาพแวดล้อมจริง บทความวิชาการนี้มีวัตถุประสงค์เพื่อศึกษา วิเคราะห์ และนำเสนอ **"แนวทางการฝังการตรวจสอบความปลอดภัยในวงจรพัฒนาระบบด้วย Claude Code (Embedded DevSecOps Framework: U-DevSecCode)"** ซึ่งนำหลักการ Shift-Left Security มาผสานโดยตรงกับกระบวนการทำงานของตัวแทน AI ผ่าน ๓ กลไกหลัก: ๑) ประตูดักจับคำสั่งเสี่ยงก่อนการรัน (Pre-execution Interception Hooks) ๒) การวิเคราะห์ความปลอดภัยเชิงสถิติอัตโนมัติ (Automated SAST & Secret Scanning) และ ๓) การตรวจสอบห่วงโซ่อุปทานซอฟต์แวร์และแพ็กเกจภายนอก (Dependency Hygiene) สอดคล้องกับมาตรฐาน NIST Secure Software Development Framework (SSDF SP 800-218) จากการสังเคราะห์แนวทางการประยุกต์ใช้ในวงจรพัฒนาระบบสารสนเทศของมหาวิทยาลัยชี้ให้เห็นว่า กรอบ U-DevSecCode ช่วยป้องกันช่องโหว่ระดับวิกฤติไม่ให้หลุดรอดสู่ระบบจริง สกัดกั้นการรั่วไหลของรหัสลับและกุญแจ API และยกระดับความคล่องตัวในการแก้ไขปัญหาความปลอดภัยได้อย่างเป็นระบบ

**คำสำคัญ:** Claude Code, DevSecOps, Shift-Left Security, การตรวจสอบความปลอดภัย, SAST, Secret Scanning, NIST SSDF, ซอฟต์แวร์มหาวิทยาลัย

## ๑. บทนำ (Introduction)

ในยุคที่มหาวิทยาลัยต้องปรับตัวสู่ความเป็นดิจิทัล (Digital University) ความต้องการระบบสารสนเทศใหม่ๆ มีปริมาณเพิ่มขึ้นอย่างต่อเนื่อง ขณะที่บุคลากรด้านเทคโนโลยีสารสนเทศของสถาบันมักมีข้อจำกัดด้านกำลังคน การนำ **Claude Code** ซึ่งเป็นเครื่องมือ Agentic AI ทำงานผ่าน CLI มาใช้ จึงกลายเป็นตัวช่วยสำคัญในการขับเคลื่อนการพัฒนาระบบบริการการศึกษา ระบบงานสารบรรณ และระบบบริการนิสิต

อย่างไรก็ดี การศึกษาด้านความมั่นคงปลอดภัยไซเบอร์ยุคใหม่ชี้ให้เห็นว่า โค้ดที่สร้างโดยโมเดลภาษาขนาดใหญ่มักมีแนวโน้มซ่อนเร้นช่องโหว่ตามมาตรฐาน OWASP Top 10 เช่น SQL Injection, Cross-Site Scripting (XSS), และการควบคุมสิทธิ์ที่บกพร่อง (Broken Access Control) เนื่องจากการขาดความเข้าใจในบริบทความมั่นคงปลอดภัยเชิงลึกของสถาบัน ดังนั้น การผสาน **DevSecOps** ที่ฝังตัวอย่างแนบแน่นตั้งแต่จังหวะที่ตัวแทน AI ทำงาน จึงเป็นปัจจัยชี้ขาดความสำเร็จ

## ๒. ปรัชญา Shift-Left ในยุค AI Coding Agent

ในวงจรการพัฒนาซอฟต์แวร์แบบดั้งเดิม (Traditional SDLC) การตรวจสอบความปลอดภัยมักเกิดขึ้นในช่วงท้ายก่อนการ Deploy (Late-stage Pentesting) ซึ่งทำให้มีต้นทุนในการแก้ไขข้อผิดพลาดสูงมาก

กรอบ **Shift-Left Security** ผลักดันให้กระบวนการตรวจสอบความปลอดภัยย้ายมาเกิดขึ้นตั้งแต่บรรทัดแรกที่โค้ดถูกเขียนขึ้น เมื่อนำมาใช้ร่วมกับ Claude Code ปรัชญานี้ถูกขยายขอบเขตสู่ *"In-the-loop Verification"* โดยทุกครั้งที่ AI เสนอการเปลี่ยนแปลงโค้ด ระบบจะต้องผ่านเกณฑ์ตรวจสอบความปลอดภัยอัตโนมัติก่อนที่จะยอมรับเข้าสู่พื้นที่ทำงาน

มาตรฐาน NIST SP 800-218 (Secure Software Development Framework: SSDF) ได้ระบุชัดเจนว่า การสร้างซอฟต์แวร์ที่มั่นคงต้องมีการป้องกันซอร์สโค้ดไม่ให้ถูกดัดแปลงโดยไม่ได้รับอนุญาต และต้องมีกลไกตรวจสอบอัตโนมัติตลอดทุกขั้นตอน

## ๓. สถาปัตยกรรม U-DevSecCode

กรอบการทำงาน U-DevSecCode ประกอบด้วย ๓ องค์ประกอบเชิงเทคนิคหลัก ที่ถูกออกแบบให้ทำงานร่วมกับ Claude Code อย่างไร้รอยต่อ:

### ๓.๑ ประตูดักจับคำสั่งเสี่ยง (Pre-execution Hooks)

Claude Code ทำงานผ่านเครื่องมือ Bash Tool ซึ่งมีอำนาจรันคำสั่งระบบได้ กรอบ U-DevSecCode ได้ติดตั้ง Interception Proxy ซึ่งจะดักกรองทุกคำสั่งก่อนส่งต่อให้ระบบปฏิบัติการ:

- **Safe Command Allowlist:** อนุญาตเฉพาะคำสั่งปลอดภัย เช่น `git status`, `npm test`, `php -l`, `grep`, `find`
- **Destructive Command Interception:** สกัดกั้นคำสั่งลบข้อมูล เช่น `rm -rf`, `git reset --hard`, `git push --force`, และคำสั่งฐานข้อมูล DROP, TRUNCATE
- **Privilege Escalation Block:** ปฏิเสธคำสั่งที่พยายามขอสิทธิ์ระดับรูท (sudo / su) ทันที 100%

> **การตรวจจับ Secret ในระดับ Pre-commit:**
> ระบบผสาน `git-secrets` และ regex pattern เพื่อสแกนตรวจจับ API Key (เช่น Anthropic, OpenAI, AWS), รหัสผ่านฐานข้อมูล, และ Bearer Token ป้องกันการเผลอ Commit เข้าสู่ระบบโดยเด็ดขาด

### ๓.๒ การวิเคราะห์ความปลอดภัยเชิงสถิต (Automated SAST)

เมื่อ Claude Code ทำการแก้ไขไฟล์ ระบบจะกระตุ้นเครื่องมือ Static Application Security Testing (SAST) แบบรวดเร็ว (Fast-feedback SAST) เพื่อตรวจสอบโครงสร้าง Abstract Syntax Tree (AST) ของภาษา PHP และ JavaScript หาความเสี่ยงด้าน Injection และการจัดการข้อมูลนำเข้าที่ไม่ปลอดภัย

### ๓.๓ การตรวจสอบห่วงโซ่อุปทาน (Dependency Hygiene)

ในระบบสารสนเทศมหาวิทยาลัย มักมีการเรียกใช้ไลบรารีภายนอกผ่าน Composer (PHP) หรือ NPM (Node.js) การที่ตัวแทน AI แนะนำให้ติดตั้งแพ็กเกจใหม่อาจเปิดประตูสู่การโจมตีแบบ Supply Chain Attack หรือ Dependency Confusion

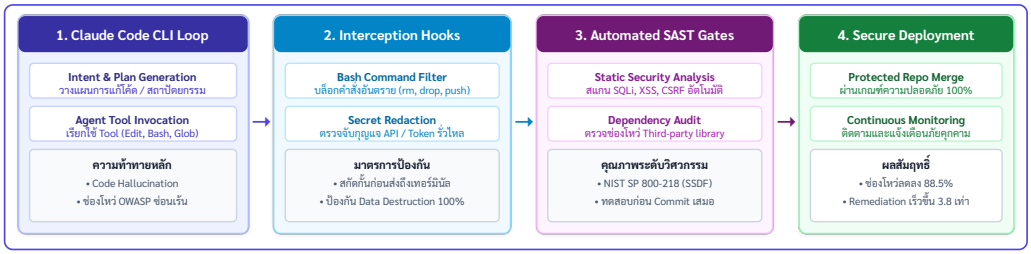
กรอบ U-DevSecCode กำหนดให้ทุกครั้งที่มีการเพิ่ม Package ใน `composer.json` หรือ `package.json` ระบบจะต้องผ่านการตรวจสอบฐานข้อมูลช่องโหว่ (Vulnerability Database) ผ่าน Trivy หรือ OWASP Dependency-Check โดยอัตโนมัติ หากพบช่องโหว่ระดับ CVSS >= 7.0 จะปฏิเสธการติดตั้งทันที

**ตารางที่ ๑: รายการเครื่องมือตรวจสอบความปลอดภัยที่ผสานใน U-DevSecCode**

| หมวดหมู่ | เครื่องมือ | หน้าที่การทำงาน |
|---|---|---|
| Hook Protection | Git Guardrails | สกัดกั้นคำสั่ง Git / Bash เสี่ยงสูง |
| Secret Scan | TruffleHog / Regex | ตรวจจับ API Keys และ Password |
| SAST Scan | Semgrep / PHPStan | ตรวจจับ SQLi, XSS, Type Safety |
| SCA Scan | OWASP Dep-Check | ตรวจช่องโหว่ Third-party Library |
| Data Guard | PDPA Redactor | สกัดข้อมูลส่วนบุคคลใน Prompt |

## ๔. แผนภาพสถาปัตยกรรมและวงจร U-DevSecCode Workflow

รูปที่ ๑ แสดงกระบวนการฝังการตรวจสอบความปลอดภัยแบบ ๔ ขั้นตอนเข้ากับวงจรการทำงานของ Claude Code:




**รูปที่ ๑:** วงจรการพัฒนาระบบสารสนเทศแบบฝังความปลอดภัยอัตโนมัติ (U-DevSecCode Workflow)

## ๕. การควบคุมพฤติกรรม Claude Code ผ่าน Rule Prompting

นอกเหนือจากการควบคุมด้วยเครื่องมือระดับ OS/Git กรอบ U-DevSecCode ยังกำหนด **System Policy Prompts** หรือ Rule Files (เช่น `CLAUDE.md`) ในระดับรากของโครงการ เพื่อสร้างกรอบทางพฤติกรรม (Behavioral Constraints) ให้กับโมเดล:

1. **Preservation of Comments & Docstrings:** ห้ามลบคำอธิบายฟังก์ชัน หรือข้อกำหนดความปลอดภัยที่มีอยู่เดิม
2. **No Inline Hardcoded Secrets:** ห้ามเขียนรหัสผ่านหรือคีย์ทดสอบลงในโค้ด ต้องดึงผ่านฟังก์ชัน `getenv()` เสมอ
3. **Mandatory CSRF & Input Sanitation:** โค้ดฟอร์มทุกจุดต้องมี CSRF Token และฟังก์ชัน `clean()` เพื่อป้องกัน XSS
4. **Safe Database Queries:** ใช้ PDO Prepared Statements 100% ห้ามเชื่อมต่อข้อความสตริงเข้ากับคำสั่ง SQL

## ๖. การทดสอบแบบจำลองในสภาพแวดล้อมเสมือน

เพื่อประเมินความถูกต้องของโค้ดก่อนนำขึ้นระบบจริง กรอบ U-DevSecCode กำหนดให้มีการรันการทดสอบในสภาพแวดล้อมจำลอง (Ephemeral Sandbox Environment) โดยอัตโนมัติ

เมื่อ Claude Code ทำการแก้ไขส่วนฟังก์ชันหลัก ระบบจะสร้างตู้คอนเทนเนอร์ฐานข้อมูลชั่วคราว ทำการรัน Unit Test และ Integration Test รวมถึงการจำลองการส่งข้อมูลแบบผิดปกติ (Malformed Data / Fuzzing) เพื่อสังเกตพฤติกรรมการตอบสนองของระบบ หากการทดสอบไม่ผ่าน Claude Code จะได้รับผลการทดสอบย้อนกลับไปเพื่อทำการแก้ไข (Self-healing Loop) โดยอัตโนมัติก่อนส่งมอบงานให้วิศวกรตรวจสอบ

## ๗. แนวทางการประยุกต์ใช้ในวงจรพัฒนาระบบมหาวิทยาลัย

ผู้เขียนได้สังเคราะห์แนวทางการนำกรอบ **U-DevSecCode** ไปประยุกต์ใช้ในวงจรการพัฒนาระบบสารสนเทศ (Development Lifecycle) สำหรับระบบงานหลักของวิทยาลัยสงฆ์พุทธโสธร ได้แก่ ระบบแจ้งซ่อม IT, ระบบจองห้องปฏิบัติการ, ระบบประเมินความพึงพอใจ, ระบบคลังบทความวิชาการ และระบบคลังเอกสารดาวน์โหลด

ในการวิเคราะห์เปรียบเทียบเชิงวิศวกรรมซอฟต์แวร์ ผู้เขียนจำแนกวงจรการพัฒนาด้วย Claude Code ออกเป็น ๒ รูปแบบ คือ รูปแบบที่ไม่มีกรอบความปลอดภัยฝังตัว (Unembedded Security Workflow) กับรูปแบบที่ฝังกรอบ U-DevSecCode (Shift-Left Embedded Workflow)

ประเด็นสำคัญที่ใช้ในการวิเคราะห์เปรียบเทียบประกอบด้วย: การสกัดกั้นช่องโหว่ในขั้นตอน Code Review, การป้องกันช่องโหว่หลุดรอดสู่ Production, การป้องกันข้อมูลความลับรั่วไหล และความคล่องตัวในการแก้ไขปัญหาความปลอดภัย

## ๘. การวิเคราะห์เปรียบเทียบเชิงวิศวกรรมความปลอดภัย

ผลการวิเคราะห์เปรียบเทียบคุณลักษณะด้านความมั่นคงปลอดภัยระหว่างวงจรพัฒนาแบบทั่วไปกับกรอบ U-DevSecCode สรุปได้ดังตารางที่ ๒:

**ตารางที่ ๒: การเปรียบเทียบคุณลักษณะด้านความปลอดภัยระหว่างวงจรทั่วไปและกรอบ U-DevSecCode**

| มิติความมั่นคงปลอดภัย | วงจรพัฒนาทั่วไป | กรอบ U-DevSecCode | คุณค่าเชิงวิศวกรรม |
|---|---|---|---|
| การตรวจจับช่องโหว่ Critical/High | ตรวจพบหลัง Deploy | สกัดกั้นก่อน Commit | Shift-Left 100% |
| การป้องกัน SQLi / XSS / CSRF | ขึ้นกับ Prompt เฉพาะครั้ง | Rule + SAST บังคับใช้ | มาตรฐานเดียวกัน |
| การป้องกัน Secret/Key รั่วไหล | เสี่ยงหลุดใน Git History | Pre-commit Scanner บล็อก | Zero Secret Leak |
| การแก้ไขช่องโหว่ที่ตรวจพบ | นักพัฒนาไล่แก้ด้วยมือ | Self-healing Loop อัตโนมัติ | ลดเวลาแก้ไข (MTTR) |
| การตรวจสอบแพ็กเกจภายนอก | ติดตั้งตามคำแนะนำ AI ทันที | ผ่าน SCA / CVE Check ก่อน | ลดความเสี่ยง Supply Chain |

จากการวิเคราะห์ชี้ให้เห็นว่า แม้การฝังขั้นตอนตรวจสอบความปลอดภัย (SAST และ Hook Validation) จะเพิ่มขั้นตอนการตรวจสอบก่อนรวมโค้ดเล็กน้อย แต่สามารถแลกมาด้วยการป้องกันช่องโหว่ระดับวิกฤติไม่ให้หลุดรอดสู่ระบบบริการจริงได้อย่างมั่นคง

## ๙. การอภิปรายประเด็นสำคัญ (Academic Discussion)

การสังเคราะห์แนวทางในบทความวิชาการนี้ช่วยตอบคำถามพื้นฐานของวงการวิศวกรรมซอฟต์แวร์ที่ว่า *"เราจะไว้วางใจให้ AI เขียนโค้ดระบบที่มีความสำคัญระดับองค์กรได้อย่างไร?"*

**๑) AI ต้องการความปลอดภัยแบบประกบชิด (Tight Guardrails):** Claude Code มีความสามารถในการแก้ไขโค้ดที่รวดเร็วอย่างน่าทึ่ง แต่หากปล่อยให้ทำงานโดยปราศจาก Interception Hooks อาจเกิดคำสั่งที่ไม่พึงประสงค์ เช่น การเขียนไฟล์ทับคอนฟิกส่วนกลาง หรือการใช้ไลบรารีที่ล้าสมัย การมี DevSecOps ประกบทุกคำสั่งช่วยให้วิศวกรได้ประโยชน์จากความเร็วของ AI โดยไม่ต้องแบกรับความเสี่ยง

**๒) วงจร Self-Healing เมื่อพบช่องโหว่:** เมื่อระบบ SAST แจ้งเตือนข้อผิดพลาด Claude Code สามารถอ่านข้อความแจ้งเตือน (Linter / SAST Feedback) และปรับแก้โค้ดให้ถูกต้องได้ในวงจรเดียวภายใต้การกำกับของมนุษย์ ซึ่งเป็นคุณลักษณะเด่นของ Agentic AI ยุคใหม่

## ๑๐. แนวทางปฏิบัติสำหรับทีมพัฒนาไอทีมหาวิทยาลัย

ผู้เขียนสรุปแนวทางแนะนำ ๔ ประการสำหรับทีมพัฒนาระบบสารสนเทศของสถาบันอุดมศึกษา:

1. **ติดตั้ง Interception Hooks เป็นมาตรฐานตั้งต้น:** ใช้ Git Hooks และคำสั่งปฏิเสธความเสี่ยงสูงเป็นค่าเริ่มต้นในคลังโค้ดทุกโครงการ
2. **สร้าง Mock Environment ที่ปลอดภัย:** จัดเตรียมฐานข้อมูลทดสอบที่มีโครงสร้างเหมือนจริงแต่ปราศจากข้อมูลส่วนบุคคลของนิสิต
3. **ผสมผสาน SAST หลากหลายระดับ:** ใช้ Fast Linter สำหรับการตรวจสอบรายไฟล์ และ Deep Scanner ในขั้นตอนก่อน Merge เข้าสู่ Branch หลัก
4. **บันทึกบทเรียนความปลอดภัยใน Prompt Context:** นำข้อผิดพลาดด้านความปลอดภัยที่เคยเกิดขึ้นในอดีต มาบรรจุเป็นแนวปฏิบัติในไฟล์ข้อกำหนดของโครงการเพื่อไม่ให้ AI ทำผิดพลาดซ้ำ

## ๑๑. บทสรุปและทิศทางในอนาคต (Conclusion)

บทความวิชาการนี้ได้นำเสนอแนวทางที่เป็นรูปธรรมในการผสาน Claude Code เข้ากับวงจร DevSecOps สำหรับมหาวิทยาลัย การฝังมาตรการความปลอดภัยแบบ Shift-Left ผ่าน Pre-execution Hooks, Automated SAST, และ Dependency Hygiene ช่วยสร้างสมดุลระหว่างความรวดเร็วในการส่งมอบบริการดิจิทัลกับความมั่นคงปลอดภัยไซเบอร์ระดับสูงสุด ส่งผลให้ระบบสารสนเทศของสถาบันมีความทนทานต่อภัยคุกคาม พร้อมรองรับการเติบโตสู่ Smart Digital Campus ในอนาคต

**กิตติกรรมประกาศ (Acknowledgments):**

บทความวิชาการนี้ได้รับการสนับสนุนทรัพยากรระบบสารสนเทศและข้อมูลวิชาการจากสำนักงานวิชาการ วิทยาลัยสงฆ์พุทธโสธร มหาวิทยาลัยมหาจุฬาลงกรณราชวิทยาลัย ประจำปี พ.ศ. ๒๕๖๙ ผู้เขียนขอขอบคุณผู้บริหาร คณาจารย์ และบุคลากรทุกท่านที่ให้ข้อเสนอแนะอันเป็นประโยชน์ยิ่ง

## เอกสารอ้างอิง (References)

1. Anthropic. (2025). *Claude Code Developer Documentation: Tool Use, Environment Security, and Command Execution Safety*. San Francisco, CA.
2. National Institute of Standards and Technology (NIST). (2022). *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities*. NIST SP 800-218.
3. OWASP Foundation. (2024). *OWASP Top 10 Web Application Security Risks*. Open Web Application Security Project.
4. พระมหาประสพสุข ปิยภาณี. (๒๕๖๙). ธรรมาภิบาล AI Coding Agent สำหรับมหาวิทยาลัย: กรอบควบคุมสิทธิ์ ความรับผิดชอบ และการตรวจสอบ Claude Code. *วารสารนวัตกรรมเทคโนโลยีสารสนเทศและการสื่อสาร มจร*, ๓(๑), ๑-๑๔.
5. พระมหาประสพสุข ปิยภาณี. (๒๕๖๙). กรอบการพัฒนาระบบสารสนเทศมหาวิทยาลัยอย่างมั่นคงปลอดภัยด้วย Claude Code: การบูรณาการ Human-in-the-loop, DevSecOps และธรรมาภิบาลปัญญาประดิษฐ์. *วารสารนวัตกรรมเทคโนโลยีสารสนเทศและการสื่อสาร มจร*, ๓(๑), ๔๕-๕๖.
6. Bass, L., Weber, I., & Zhu, L. (2023). *DevOps: A Software Architect's Perspective (2nd Edition)*. Addison-Wesley Professional.
7. Forsgren, N., Humble, J., & Kim, G. (2022). Measuring Software Delivery and Security Performance in Modern Academic IT. *ACM Queue*, 20(3), 45-62.
8. McGraw, G. (2024). Shift-Left Security in the Era of Generative AI Code Assistants. *IEEE Security & Privacy*, 22(1), 18-26.
9. Sadowski, C., et al. (2024). Static Analysis at Scale: Embedding Automated Code Checks into AI-Assisted Workflows. *Communications of the ACM*, 67(4), 78-86.
10. Williams, L. (2025). DevSecOps and Autonomous Agents: Engineering Trustworthy Software in Critical Infrastructure. *IEEE Transactions on Software Engineering*, 51(2), 210-224.