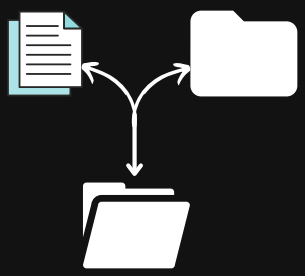
Hakka.
ハッカー

SQL Injection

แบบสอน SQL

นำเสนอโดย
Hakka.
ハッカー

การจำแนกประเภทของการโจมตี และ การตอบโต้ ด้วย SQL Injection



ABSTRACT

การโจมตี ด้วย SQL Injection ก่อให้เหิดภัยคุกคามด้านความปลอดภัยอย่างร้ายแรง ต่อ เว็บแอปพลิเคชั่น : อนุญาตให้ผู้โจมตีเข้าถึงฐานข้อมูลที่อยู่ภายใต้ Applications และ ข้อมูลที่ละเอียดอ่อน ซึ่งฐานข้อมูลเหล่าหนี้ มีได้โดยไม่จำกัด แม้ว่านักวิจัย และ ผู้ปฏิบัติงานได้เสนอวิธีการต่างๆ เพื่อแก้ไขปัญหา SQL Injection, แนวทางปัจจุบันไม่สามารถระบุขอบเขตทั้งหมดของปัญหาได้ หรือ มีข้อจำกัดที่ขัดขวางการใช้งาน และ การยอมรับ นักวิจัย และ ผู้ปฏิบัติงานหลายคนคุ้นเคยกับแค่เพียงชุดยย่อยของเทคนิคที่หลากหลาย สำหรับผู้โจมตีที่พยายามใช้ประโยชน์จากช่องโหว่ของ SQL Injection ด้วยเหตุนี้ โซลูชันจำนวนมากที่เสนอในเอกสารจึงกล่าวถึงปัญหาบางอย่างที่เกี่ยวข้องกับ SQL Injection เท่านั้น เพื่อแก้ไขปัญหานี้ เราขอนำเสนอการตรวจสอบอย่างละเอียดเกี่ยวกับการโจมตีแบบ SQL Injection ประเภทต่องๆ ที่รู้จักในปัจจุบัน สำหรับการโจมตีแต่ละประเภท เรามีคำอธิบาย และ ตัวอย่างวิธีการโจมตีประเภทนั้น นอกจากนี้ เรายังนำเสนอ และ วิเคราะห์เทคนิคการตรวจจับ และ ป้องกันที่มีอยู่จากการโจมตีด้วย SQL Injection สำหรับแต่ละเทคนิค เราจะพูดถึงจุดแข็ง และ จุดอ่อนของเทคนิคนี้ในการจัดการกับการโจมตีด้วย SQL Injection ทั้งหมด

1.INTRODUCTION(บทนำ)

ช่องโหว่ของ SQL Injection ได้รับการอธิบายไว้ ว่า เป็นหนึ่งในภัยคุกคามที่ร้ายแรงที่สุด สำหรับ เว็บแอปพลิเคชั่น เว็บแอปพลิเคชั่น ที่เสี่ยงต่อ SQL Injection อาจทำให้ผู้โจมตีสามารถเข้าถึงฐานข้อมูลพื้นฐานของตนได้อย่างสมบูรณ์ เนื่องจากฐานข้อมูลของผู้บริโภค หรือ ผู้ใช้ที่มีความละเอียดอ่อนการละเมิดความปลอดภัยที่ตามมาอาจรวมถึงการขโมยข้อมูลประจำตัว การสูญหายของข้อมูลที่เป็นความลับ และ การฉ้อโกง ในบางกรณี ผู้โจมตีสามารถใช้ช่องโหว่ของ SQL Injection เพื่อควบคุม และ ทำให้ระบบที่โฮสต์เว็บแอปพลิเคชั่นเสียหาย เว็บแอปพลิเคชั่นที่เสี่ยงต่อการโจมตีด้วย SQL Injection (SQLIA) เป็นที่แพร่หลาย การศึกษาโดย Gartner Group บนเว็บไซต์อินเทอร์เน็ตกว่า 300 แห่ง แสดงให้เห็นว่าส่วนใหญ่อาจเสี่ยงต่อ SQLIA อันที่จริง SQLIA ประสบความสำเร็จในการกำหนดเป้าหมายผู้ที่ตกเป็นเหยื่อระดับสูง เช่น Travelocity, FTD.com และ GUESS Inc

SQL Injection หมายถึง คลาสของการโจมตีด้วย code-injection ซึ่งข้อมูลที่ผู้ใช้ให้มารวมอยู่ในการสืบค้น SQL ในลักษณะที่ส่วนหนึ่งของอินพุตของผู้ใช้จะถือว่าเป็น SQL code โดยใช้ประโยชน์จากช่องโหว่เหล่านี้ ผู้โจมตีสามารถส่งคำสั่ง SQL ไปยังฐานข้อมูลได้โดยตรง การโจมตีเหล่านี้เป็นภัยคุกคามร้ายแรงต่อเว็บแอปพลิเคชั่นใดๆ ที่รับข้อมูลจากผู้ใช้ และ รวมเข้ากับการสืบค้น SQL ไปยังฐานข้อมูลพื้นฐาน เว็บแอปพลิเคชั่นส่วนใหญ่

สาเหตุของช่องโหว่ SQL Injection นั้นค่อนข้างง่าย และ เข้าใจง่าย : การตรวจสอบอินพุตของผู้ใช้ไม่เพียงพอ เพื่อแก้ไขปัญหานี้ นักพัฒนาซอฟต์แวร์ได้เสนอแนวทางการเข้ารหัสที่หลากหลาย ที่ส่งเสริมแนวปฏิบัติในการเข้ารหัสเชิงป้องกัน เช่น การเข้ารหัสอินพุตของผู้ใช้ และ การตรวจสอบความถูกต้อง การใช้เทคนิคเหล่านี้อย่างเข้มงวด และ เป็นระบบ เป็นวิธีแก้ปัญหาที่มีประสิทธิภาพในการป้องกันช่องโหว่ของ SQL Injection อย่างไรก็ตาม ในทางปฏิบัติ การนำเทคนิคดังกล่าวไปใช้นั้นอิงจากฝีมือมนุษย์ ดังนั้นจึงมีแนวโน้มที่จะเกิดข้อผิดพลาด นอกจากนี้ การแก้ไขฐานรหัสดั้งเดิมที่อาจมีช่องโหว่ของ SQL Injection อาจเป็นงานที่ต้องใช้แรงงานมาก

แม้ว่าเมื่อเร็วๆนี้ ได้มีการให้ความสนใจอย่างมากกับปัญหาช่องโหว่ของ SQL Injection แต่โซลูชันที่เสนอจำนวนมากไม่สามารถจัดการกับปัญหาทั้งหมดได้ มี SQLIA หลายประเภท และ รูปแบบพื้นฐานเหล่านี้นับไม่ถ้วน นักวิจัย และ ผู้ปฏิบัติงานมักไม่ทราบถึงเทคนิคต่างๆมากมาย ที่สามารถใช้เพื่อดำเนินการ SQLIA

ดังนั้น โซลูชันส่วนใหญ่ที่เสนอให้ตรวจจับ หรือ ป้องกันเฉพาะชุดย่อยของ SQLIA ที่เป็นไปได้ เพื่อแก้ไขปัญหานี้ เราขอนำเสนอแบบสำรวจที่ครอบคลุมเกี่ยวกับการโจมตีด้วย SQL Injection ที่ทราบจนถึงปัจจุบัน ในการรวบรวมแบบสำรวจ เราใช้ข้อมูลที่รวบรวมจากแหล่งต่างๆ เช่น เอกสาร เว็บไซต์ รายชื่อผู้รับจดหมาย และ ผู้เชี่ยวชาญในพื้นที่ สำหรับการโจมตีแต่ละประเภทที่พิจารณา เราจะให้ลักษณะของการโจมตี แสดงผลกระทบ และ ให้ตัวอย่างวิธีการโจมตีประเภทนั้น จากนั้นใช้ชุดประเภทการโจมตีเพื่อประเมินเทคนิคการตรวจจับ และ ป้องกันที่ทันสมัย และ เปรียบเทียบจุดแข็ง และ จุดอ่อน ผลการเปรียบเทียบนี้ แสดงให้เห็นถึงประสิทธิภาพของเทคนิคเหล่านี้

ส่วนที่เหลือของเรื่องนี้ ได้รับการจัดระเบียบดังนี้ : ส่วนที่ 2 ให้ข้อมูลพื้นฐานเกี่ยวกับ SQLIA และ แนวคิดที่เกี่ยวข้อง ส่วนที่ 4 กำหนด และ นำเสนอประเภทการโจมตีที่แตกต่างกัน ส่วนที่ 5 และ 6 ทบทวน และ ประเมินเทคนิคปัจจุบันเทียบกับ SQLIA สุดท้ายนี้ เรามีบทสรุป และ ข้อสรุปในส่วนที่ 7

2.BACKGROUND ON SQLIAS(ความเป็นมาเกี่ยวกับ SQLIAS)

อย่างที่รู้ว่า SQL Injection Attack (SQLIA) เกิดขึ้นเมื่อผู้โจมตีเปลี่ยนผลกระทบที่ตั้งใจไว้ของการสืบค้น SQL โดยการแทรกคำสำคัญ หรือ ตัวดำเนินการ SQL ใหม่ลงในแบบสอบถาม คำจำกัดความที่ไม่เป็นทางการนี้ มีวัตถุประสงค์เพื่อรวมตัวแปรทั้งหมดของ SQLIA ที่รายงานในวรรณกรรม และ นำเสนอในเรื่องนี้ สำหรับคำจำกัดความที่เป็นทางการของ SQLIA ในส่วนที่เหลือของส่วนนี้ เรากำหนดลักษณะสำคัญสองประการของ SQLIA ที่เราใช้เพื่ออธิบายการโจมตี : กลไกล Injection และ เจตนาในการโจมตี

2.1Injection Mechanisms

คำสั่ง SQL ที่เป็นอันตรายสามารถนำไปใช้กับแอปพลิเคชั่นที่มีช่องโหว่ได้โดยใช้กลไกการป้องกันข้อมูลที่แตกต่างกันมากมาย ในส่วนนี้ เราจะอธิบายกลไกที่พบบ่อยที่สุด

Injection through user input : ในกรณีนี้ ผู้โจมตีจะป้อนคำสั่ง SQL โดยการป้อนข้อมูลของผู้ใช้ที่ออกแบบมาอย่างเหมาะสม เว็บแอปพลิเคชันสามารถอ่านอินพุตของผู้ใช้ได้หลายวิธีตามสภาพแวดล้อมที่แอปพลิเคชันถูกปรับใช้ ใน SQLIA ส่วนใหญ่ที่กำหนดเป้าหมายเว็บแอปพลิเคชัน อินพุตของผู้ใช้มักมาจากการส่งแบบฟอร์มที่ส่งไปยังเว็บแอปพลิเคชันผ่านคำขอ HTTP GET หรือ POST โดยทั่วไปแล้ว เว็บแอปพลิเคชันจะสามารถเข้าถึงอินพุตของผู้ใช้ที่อยู่ในคำขอเหล่านี้ได้ เช่นเดียวกับการเข้าถึงตัวแปรอื่นๆ ในสภาพแวดล้อม

Injection through cookies : cookies คือ ไฟล์ที่มีข้อมูลสถานะที่สร้างโดยเว็บแอปพลิเคชัน และ จัดเก็บไว้ในเครื่องไคลเอนต์ เมื่อไคลเอนต์กลับมาที่เว็บแอปพลิเคชันสามารถใช้ cookies เพื่อกู้คืนข้อมูลสถานะของไคลเอนต์ได้ เนื่องจากไคลเอนต์สามารถควบคุมการจัดเก็บ cookies ได้ ไคลเอนต์ที่เป็นอันตรายจึงสามารถยุ่งเกี่ยวกับเนื้อหาของ cookies ได้ หากเว็บแอปพลิเคชันใช้เนื้อหาของ cookies เพื่อสร้างแบบสอบถาม SQL ผู้โจมตีสามารถส่งการโจมตีได้อย่างง่ายดายโดยฝังไว้ใน cookies

Injection through server variables : ตัวแปรเซิร์ฟเวอร์ คือ ชุดของตัวแปรที่มี HTTP ส่วนหัวของเครือข่าย และ ตัวแปรด้านสิ่งแวดล้อม เว็บแอปพลิเคชันใช้ตัวแปรเซิร์ฟเวอร์เหล่านี้ในหลากหลายวิธี เช่น การบันทึกสถิติการใช้งาน และ การระบุแนวโน้มการท่องเว็บ หากตัวแปรเหล่านี้ถูกบันทึกลงในฐานข้อมูลโดยไม่มีการฆ่าเชื้อ อาจทำให้เกิดช่องโหว่ SQL Injection เนื่องจากผู้โจมตีสามารถปลอมแปลงค่าที่วางไว้ใน HTTP และ ส่วนหัวของเครือข่าย พวกเขาจึงสามารถใช้ประโยชน์จากช่องโหว่นี้โดยการวาง SQLIA ลงในส่วนหัวโดยตรง เมื่อแบบสอบถามเพื่อบันทึกตัวแปรเซิร์ฟเวอร์ถูกส่งไปยังฐานข้อมูล การโจมตีในส่วนหัวที่ปลอมแปลงจะถูกทริกเกอร์

อย่างที่รู้ว่า SQL Injection Attack (SQLIA) เกิดขึ้นเมื่อผู้โจมตีเปลี่ยนผลกระทบที่ตั้งใจไว้ของการสืบค้น SQL โดยการแทรกคำสำคัญ หรือ ตัวดำเนินการ SQL ใหม่ลงในแบบสอบถาม คำจำกัดความที่ไม่เป็นทางการนี้ มีวัตถุประสงค์เพื่อรวมตัวแปรทั้งหมดของ SQLIA ที่รายงานในวรรณกรรม และ นำเสนอในเรื่องนี้ สำหรับคำจำกัดความที่เป็นทางการของ SQLIA ในส่วนที่เหลือของส่วนนี้ เรากำหนดลักษณะสำคัญสองประการของ SQLIA ที่เราใช้เพื่ออธิบายการโจมตี : กลไกล Injection และ เจตนาในการโจมตี

Second-order injection : ในการแทรกลำดับที่ 2 ผู้โจมตีจะทำการป้อนข้อมูลที่เป็นอันตรายลงในระบบ หรือ ฐานข้อมูลเพื่อทริกเกอร์ SQLIA ทางอ้อม เมื่อมีการใช้งานอินพุตนั้นในภายหลัง วัตถุประสงค์ของการโจมตีประเภทนี้แตกต่างอย่างมากจากการโจมตีแบบ injection (เช่น firstorder) Second-order injection จะไม่พยายามทำให้เกิดการโจมตี เมื่ออินพุตที่เป็นอันตรายถึงฐานข้อมูลในขั้นต้น ผู้โจมตีต้องอาศัยความรู้ ว่า ข้อมูลที่ป้องกันจะถูกนำไปใช้ที่ใดในภายหลัง และ สร้างการโจมตีเพื่อให้เกิดขึ้นระหว่างการใช้งานนั้น เพื่อความกระจ่าง เราได้นำเสนอตัวอย่างคลาสสิกของการโจมตีด้วย Second order injection ในตัวอย่าง ผู้ใช้ลงทะเบียนบนเว็บไซต์โดยใช้ชื่อผู้ใช้เริ่มต้น เช่น "admin’ -- " แอปพลิเคชันหลีกเลี่ยงเครื่องหมายคำพูดเดียวในอินพุตอย่างถูกต้องก่อนที่จะจัดเก็บไว้ในฐานข้อมูล เพื่อป้องกันผลกระทบที่อาจเป็นอันตราย ณ จุดนี้ ผู้ใช้แก้ไขรหัสผ่านของตน ซึ่งเป็นการดำเนินการที่มักเกี่ยวข้องกับ (1)การตรวจสอบว่าผู้ใช้ทราบรหัสผ่านปัจจุบัน และ (2)เปลี่ยนรหัสผ่านหากการตรวจสอบสำเร็จ เมื่อต้องการทำเช่นนี้ โปรแกรมประยุกต์บนเว็บอาจสร้างคำสั่ง SQL ดังต่อไปนี้ :

```
queryString="UPDATE users SET password=' " + newPassword +
" ' WHERE userName=' " + userName + " ' AND password=' " +
oldPassword + " ' "
```

newPassword และ oldPassword เป็นรหัสผ่านใหม่ และ รหัสผ่านเก่า ตามลำดับ และ ชื่อผู้ใช้ คือ ชื่อของผู้ใช้ที่เข้าสู่ระบบในปัจจุบัน (เช่น "admin'--") ดังนั้น สตริงการสืบค้นที่ส่งไปยังฐานข้อมูลคือ(สมมติว่า newPassword และ oldPassword คือ "newpwd" และ "oldpwd") :

```
UPDATE users SET password=' newpwd'
WHERE usersName= ' admin' --' AND password=' oldpwd'
```

เนื่องจาก "--" เป็นโอเปอเรเตอร์ความคิดเห็นของ SQL ทุกอย่างที่ตามมา คือ ละเว้นโดยฐานข้อมูล ดังนั้น ผลลัพธ์ของแบบสอบถามนี้ คือ ฐานข้อมูลเปลี่ยนรหัสผ่านของผู้ดูแลระบบ("admin") เป็นค่าที่ผู้โจมตีกำหนด

Second-order injection อาจเป็นเรื่องยากโดยเฉพาะอย่างยิ่งในการตรวจจับ และ ป้องกัน เนื่องจากจุดที่ injection แตกต่างจากจุดที่การโจมตีปรากฏขึ้นจริง นักพัฒนาซอฟต์แวร์อาจหลีกเลี่ยง ตรวจสอบการพิมพ์ และ กรองข้อมูลที่มาจากผู้ใช้อย่างเหมาะสม และ ถือว่าปลอดภัย ต่อมา เมื่อใช้ข้อมูลนั้นในบริบทที่แตกต่างกัน หรือ เพื่อสร้างการสืบค้นข้อมูลประเภทอื่น การป้อนข้อมูลที่ถูกล้างก่อนหน้านี้อาจส่งผลให้เกิดการโจมตีแบบ injection

2.2Attack Intent

การโจมตียังสามารถกำหนดลักษณะตามเป้าหมาย หรือ เจตนาของผู้โจมตี ดังนั้น คำจำกัดความประเภทการโจมตี แต่ละประเภทที่เราให้ไว้ในส่วนที่ 4 จึงรวมรายการของความตั้งใจในการโจมตีตั้งแต่หนึ่งรายการขึ้นไปที่กำหนดไว้ในส่วนนี้

Identifying injectable parameters : ผู้โจมตีต้องการตรวจสอบเว็บแอปพลิเคชันเพื่อค้นหาว่าพารามิเตอร์ใด และ ช่องป้อนข้อมูลของผู้ใช้ใดที่เสี่ยงต่อ SQLIA

Performing database finger-printing : ผู้โจมตีต้องการค้นหาประเภทและ เวอร์ชันของฐานข้อมูลที่เว็บแอปพลิเคชันใช้อยู่ ฐานข้อมูลบางประเภทตอบสนองต่อการสืบค้น และ การโจมตีที่แตกต่างกัน และ ข้อมูลนี้สามารถใช้เพื่อ "ฟingerprint" ฐานข้อมูลได้ การทราบประเภท และ เวอร์ชันของฐานข้อมูลที่ใช้โดยเว็บแอปพลิเคชันช่วยให้ผู้โจมตีสามารถสร้างการโจมตีเฉพาะฐานข้อมูลได้

Determining database schema : ในการดึงข้อมูลจากฐานข้อมูลอย่างถูกต้อง ผู้โจมตีมักจะจำเป็นต้องทราบข้อมูลสคีมาของฐานข้อมูล เช่น ชื่อตาราง ชื่อคอลัมน์ และ ประเภทข้อมูลคอลัมน์ การโจมตีโดยเจตนานี้ถูกสร้างขึ้นเพื่อรวบรวม หรือ อนุมานข้อมูลประเภทนี้

Extracting data : การโจมตีประเภทนี้ใช้เทคนิคที่จะดึงค่าข้อมูลจากฐานข้อมูล ข้อมูลนี้อาจมีความละเอียดอ่อน และ เป็นที่ต้องการอย่างมากสำหรับผู้โจมตี ทั้งนี้ ขึ้นอยู่กับประเภทของเว็บแอปพลิเคชัน การโจมตีโดยเจตนานี้เป็นประเภท SQLIA ที่พบบ่อยที่สุด

Adding or modifying data : เป้าหมายของการโจมตีเหล่านี้ คือ การเพิ่ม หรือ เปลี่ยนแปลงข้อมูลในฐานข้อมูล

Performing denial of service : การโจมตีเหล่านี้ดำเนินการเพื่อปิดฐานข้อมูลของเว็บแอปพลิเคชัน ดังนั้นจึงเป็นการปฏิเสธบริการสำหรับผู้ใช้รายอื่น การโจมตีที่เกี่ยวข้องกับการล็อก หรือ วางตารางฐานข้อมูลก็จัดอยู่ในหมวดหมู่นี้เช่นกัน

Evading detection : หมวดหมู่นี้หมายถึงเทคนิคการโจมตีบางอย่างที่ใช้เพื่อหลีกเลี่ยงการตรวจสอบ และ การตรวจจับโดยกลไกการป้องกันระบบ

Bypassing authentication : เป้าหมายของการโจมตีประเภทนี้คือการอนุญาตให้ผู้โจมตีสามารถเลี่ยงผ่านกลไกการตรวจสอบฐานข้อมูล และแอปพลิเคชัน การข้ามกลไกดังกล่าวอาจทำให้ผู้โจมตีสามารถสันนิษฐานถึงสิทธิ์ และ สิทธิพิเศษที่เกี่ยวข้องกับผู้ใช้แอปพลิเคชันรายอื่น

Executing remote commands : การโจมตีประเภทนี้พยายามรันคำสั่งโดยเด็ดขาดบนฐานข้อมูล คำสั่งเหล่านี้สามารถจัดเก็บโพรซีเดอร์ หรือ ฟังก์ชันที่ผู้ใช้ฐานข้อมูลใช้ได้

Performing privilege escalation : การโจมตีเหล่านี้ใช้ประโยชน์จากข้อผิดพลาดในการใช้งาน หรือ ตรรกะในฐานข้อมูลเพื่อยกระดับสิทธิ์ของผู้โจมตี ในทางตรงกันข้ามกับการเลี่ยงการโจมตีการตรวจสอบความถูกต้อง การโจมตีเหล่านี้มุ่งเน้นไปที่การใช้ประโยชน์จากสิทธิ์ของผู้ใช้ฐานข้อมูล

3.EXAMPLE APPLICATION(แอปพลิเคชันตัวอย่าง)

ก่อนที่จะพูดถึงการโจมตีประเภทต่างๆ เราขอแนะนำแอปพลิเคชันตัวอย่างที่มีช่องโหว่ SQL Injection เราใช้ตัวอย่างนี้ในหัวข้อถัดไปเพื่อให้ตัวอย่างการโจมตี

```
1. String login, password, pin, query
2. login = getParameter("login");
3. password = getParameter("pass");
4. pin = getParameter("pin");
5. Connection conn.createConnection("MyDataBase");
6. query = "SELECT accounts FROM users WHERE login='" +
7.     login + "' AND pass='" + password +
8.     "' AND pin=" + pin;
9. ResultSet result = conn.executeQuery(query);
10. if (result!=NULL)
11.     displayAccounts(result);
12. else
13.     displayAuthFailed();
```

รูปที่ 1: ข้อความที่ตัดตอนมาของการนำเซิร์ฟเล็ตไปใช้

โปรดทราบว่าตัวอย่างอ้างถึงช่องโหว่ที่ค่อนข้างง่ายซึ่งสามารถป้องกันได้โดยใช้การแก้ไขการเข้ารหัสที่ตรงไปตรงมา เราใช้ตัวอย่างนี้เพื่อจุดประสงค์ในการอธิบายเท่านั้น เนื่องจากเข้าใจได้ง่ายและเป็นภาพรวมเพียงพอที่จะแสดงให้เห็นการโจมตีประเภทต่างๆ

โค้ดที่ตัดตอนมาในรูปที่ 1 ใช้ฟังก์ชันการเข้าสู่ระบบสำหรับแอปพลิเคชัน มันขึ้นอยู่กับการใช้งานฟังก์ชั่นการเข้าสู่ระบบที่คล้ายคลึงกันที่เราพบในแอปพลิเคชันบนเว็บที่มีอยู่ โค้ดในตัวอย่างใช้พารามิเตอร์อินพุต login, pass, and pin เพื่อสร้าง SQL query แบบไดนามิกและส่งไปยังฐานข้อมูล

ตัวอย่างเช่น หากผู้ใช้ส่งข้อมูลการเข้าสู่ระบบ รหัสผ่าน และ pin เป็น "doe" "secret" และ "123" แอปพลิเคชันจะสร้างและส่งข้อความค้นหาแบบไดนามิก :

```
SELECT accounts FROM users WHERE
login='doe' AND pass='secret' AND pin=123
```

หากล็อกอิน รหัสผ่าน และพินตรงกับรายการที่เกี่ยวข้องในฐานข้อมูล ข้อมูลบัญชี "doe's" จะถูกส่งคืนและแสดงด้วยฟังก์ชัน displayAccounts() หากไม่มีข้อมูลที่ตรงกันในฐานข้อมูล ฟังก์ชัน displayAuthFailed() จะแสดงข้อความแสดงข้อผิดพลาดที่เหมาะสม

4.SQLIA TYPES(ประเภท SQLIA)

ในส่วนนี้ เราจะนำเสนอและหารือเกี่ยวกับ SQLIA ประเภทต่างๆ ที่รู้จักกันในปัจจุบัน สำหรับการโจมตีแต่ละประเภท เราให้ชื่อที่สื่อความหมาย เจตนาในการโจมตีตั้งแต่หนึ่งรายการขึ้นไป คำอธิบายของการโจมตี ตัวอย่างการโจมตี และชุดข้อมูลอ้างอิงถึงสิ่งตีพิมพ์และเว็บไซต์ที่กล่าวถึงเทคนิคการโจมตีและรูปแบบต่างๆ โดยละเอียดมากขึ้น

โดยทั่วไปแล้ว การโจมตีประเภทต่างๆ จะไม่ทำแบบแยกส่วน หลายอย่างใช้ร่วมกันหรือตามลำดับ ขึ้นอยู่กับเป้าหมายเฉพาะของผู้โจมตี โปรดทราบว่ามีรูปแบบการโจมตีแต่ละประเภทนับไม่ถ้วน เราไม่ได้นำเสนอรูปแบบการโจมตีที่เป็นไปได้ทั้งหมด แต่นำเสนอตัวอย่างที่เป็นตัวแทนเพียงตัวอย่างเดียว

Tautology

Attack Intent : ข้ามการตรวจสอบ การระบุพารามิเตอร์ที่ inject ได้ การดึงข้อมูล

Description : เป้าหมายทั่วไปของการโจมตีแบบ Tautology คือการ inject โค้ดในข้อความสั่งแบบมีเงื่อนไขอย่างน้อยหนึ่งคำสั่ง เพื่อให้พวกเขาประเมินว่าเป็นจริงเสมอ ผลที่ตามมาของการโจมตีนี้ขึ้นอยู่กับวิธีการใช้ผลลัพธ์ของ query ภายในแอปพลิเคชัน การใช้งานทั่วไปส่วนใหญ่คือการเลี่ยงผ่านหน้าการรับรองความถูกต้องและดึงข้อมูล ในการฉีดประเภทนี้ ผู้โจมตีจะใช้ประโยชน์จากฟิลด์ที่ฉีดได้ซึ่งใช้ในเงื่อนไข "WHERE" ของเคียวรี การแปลงเงื่อนไขเป็น Tautology ทำให้แถวทั้งหมดในตารางฐานข้อมูลที่กำหนดเป้าหมายโดยแบบสอบถามถูกส่งกลับ โดยทั่วไป เพื่อให้การโจมตีแบบ tautology ทำงานได้ ผู้โจมตีต้องพิจารณาไม่เพียงแต่พารามิเตอร์ที่ inject ได้/ช่องโหว่เท่านั้น แต่ยังรวมถึงโครงสร้างการเข้ารหัสที่ประเมินผลลัพธ์ของแบบสอบถามด้วย โดยทั่วไป การโจมตีจะสำเร็จเมื่อรหัสแสดงระเบียนที่ส่งคืนทั้งหมดหรือดำเนินการบางอย่างหากมีการส่งคืนระเบียนอย่างน้อยหนึ่งรายการ

Example : ในตัวอย่างนี้โจมตี ผู้โจมตีส่ง " ' หรือ 1=1 – " สำหรับช่องป้อนข้อมูลการเข้าสู่ระบบ(ข้อมูลที่ส่งสำหรับฟิลด์อื่นไม่เกี่ยวข้อง) query ที่ได้คือ :

```
SELECT accounts FROM users WHERE
login='' or 1=1 -- AND pass='' AND pin=
```

โค้ดที่ inject เข้าไปในเงื่อนไข (OR 1=1) จะเปลี่ยน WHERE clause ทั้งหมดให้เป็น Tautology ฐานข้อมูลใช้เงื่อนไขเป็นพื้นฐานสำหรับการประเมินแต่ละแถว และตัดสินใจว่าจะกลับไปที่แอปพลิเคชันใด เนื่องจากเงื่อนไขเป็น Tautology query จึงประเมินว่าเป็นจริงสำหรับแต่ละแถวในตารางและส่งคืนทั้งหมด ในตัวอย่างของเรา ชุดที่ส่งคืนจะประเมินเป็นค่าที่ไม่เป็นค่าว่าง ซึ่งทำให้แอปพลิเคชันสรุปว่าการพิสูจน์ตัวตนผู้ใช้ประสบความสำเร็จ
ดังนั้น แอปพลิเคชันจะเรียกใช้เมธอด displayAccounts() และแสดงบัญชีทั้งหมดในชุดที่ส่งคืนโดยฐานข้อมูล

Illegal/Logically Incorrect Queries

Attack Intent : การระบุพารามิเตอร์ที่ inject ได้ การพิมพ์ลายนิ้วมือของฐานข้อมูล การดึงข้อมูล

Description : การโจมตีนี้ช่วยให้ผู้โจมตีรวบรวมข้อมูลสำคัญเกี่ยวกับประเภทและ โครงสร้างของฐานข้อมูลส่วนหลังของเว็บแอปพลิเคชัน การโจมตีถือเป็นขั้นตอนเบื้องต้นในการรวบรวมข้อมูลสำหรับการโจมตีอื่นๆ ช่องโหว่ที่เกิดจากการโจมตีนี้ คือ หน้าข้อผิดพลาดเริ่มต้นที่ส่งคืนโดยเซิร์ฟเวอร์แอปพลิเคชันมักจะอธิบายมากเกินไป อันที่จริง ข้อเท็จจริงง่ายๆ ที่ข้อความแสดงข้อผิดพลาดถูกสร้างขึ้นมักจะเปิดเผยพารามิเตอร์ที่เปราะบาง / สามารถฉีดได้ต่อผู้โจมตี ข้อมูลข้อผิดพลาดเพิ่มเติม ซึ่งเดิมมีวัตถุประสงค์เพื่อช่วยโปรแกรมเมอร์ดีบักแอปพลิเคชันของตน ช่วยให้ผู้โจมตีได้รับข้อมูลเกี่ยวกับสคีมาของฐานข้อมูลส่วน back-end เมื่อทำการโจมตีนี้ ผู้โจมตีจะพยายามใส่ข้อความสั่งที่ทำให้เกิดไวยากรณ์ การแปลงประเภท หรือข้อผิดพลาดทางตรรกะในฐานข้อมูล ข้อผิดพลาดทางไวยากรณ์สามารถใช้เพื่อระบุพารามิเตอร์ที่ inject ได้ ข้อผิดพลาดประเภทสามารถใช้เพื่ออนุมานชนิดข้อมูลของคอลัมน์บางคอลัมน์หรือเพื่อดึงข้อมูล ข้อผิดพลาดทางตรรกะมักจะเปิดเผยชื่อของตารางและคอลัมน์ที่ทำให้เกิดข้อผิดพลาด

Example : เป้าหมายของการโจมตีตัวอย่างนี้คือทำให้เกิดข้อผิดพลาดในการแปลงประเภทที่สามารถเปิดเผยข้อมูลที่เกี่ยวข้องได้ ในการดำเนินการนี้ ผู้โจมตีจะแทรกข้อความต่อไปนี้ลงในฟิลด์อินพุต : "convert(int,(select top 1 name from sysobjects where xtype='u'))" แบบสอบถามที่ได้คือ :

```
SELECT accounts FROM users WHERE login='' AND
pass='' AND pin= convert (int,(select top 1 name from
sysobjects where xtype='u'))
```

ในสตริงการโจมตี แบบสอบถามแบบใช้เลือกข้อมูลแบบ inject จะพยายามแยกตารางผู้ใช้แรก (xtype='u') จากตารางข้อมูลเมตาของฐานข้อมูล (สมมติว่าแอปพลิเคชันใช้ Microsoft SQL Server ซึ่งตารางข้อมูลเมตาเรียกว่า sysobjects) แบบสอบถามจะพยายามแปลงชื่อตารางนี้เป็นจำนวนเต็ม เนื่องจากนี่ไม่ใช่การแปลงประเภททางกฎหมาย ฐานข้อมูลจึงแสดงข้อผิดพลาด
เนื่องจากนี่ไม่ใช่การแปลงประเภททางกฎหมาย ฐานข้อมูลจึงแสดงข้อผิดพลาด
สำหรับ Microsoft SQL Server *ข้อผิดพลาดจะเป็น : "Microsoft OLE DB Provider for SQL Server (0x80040E07) เกิดข้อผิดพลาดในการแปลงค่า nvarchar 'CreditCards' เป็นคอลัมน์ประเภทข้อมูล int"*
มีข้อมูลที่เป็นประโยชน์สองชิ้นในข้อความนี้ที่ช่วยผู้โจมตี ประการแรก ผู้โจมตีจะเห็นว่าฐานข้อมูลนั้นเป็นฐานข้อมูล SQL Server เนื่องจากข้อความแสดงข้อผิดพลาดได้ระบุข้อเท็จจริงนี้ไว้อย่างชัดเจน ประการที่สอง ข้อความแสดงข้อผิดพลาดแสดงค่าของสตริงที่ทำให้เกิดการแปลงประเภท ในกรณีนี้ ค่านี้ยังเป็นชื่อของตารางแรกที่ผู้ใช้กำหนดในฐานข้อมูล : "CreditCards" สามารถใช้กลยุทธ์ที่คล้ายกันเพื่อแยกชื่อและประเภทของแต่ละคอลัมน์ในฐานข้อมูลอย่างเป็นระบบ การใช้ข้อมูลนี้เกี่ยวกับสคีมาของฐานข้อมูล ผู้โจมตีสามารถสร้างการโจมตีเพิ่มเติมโดยมุ่งเป้าไปที่ข้อมูลเฉพาะ

Union Query

Attack Intent : ข้ามการตรวจสอบสิทธิ์, ดึงข้อมูล
Description : ในการโจมตีแบบ union-query ผู้โจมตีใช้ประโยชน์จากพารามิเตอร์ที่มีช่องโหว่เพื่อเปลี่ยนชุดข้อมูลที่ส่งคืนสำหรับการสืบค้นที่กำหนด ด้วยเทคนิคนี้ ผู้โจมตีสามารถหลอกให้แอปพลิเคชันส่งคืนข้อมูลจากตารางที่แตกต่างจากที่นักพัฒนาตั้งใจไว้ ผู้โจมตีทำได้โดยใส่คำสั่งของแบบฟอร์ม :
UNION SELECT <rest of injected query>+
เนื่องจากผู้โจมตีควบคุมการสืบค้นที่สอง / แทรกเข้าไปอย่างสมบูรณ์ พวกเขาจึงสามารถใช้การสืบค้นนั้นเพื่อดึงข้อมูลจากตารางที่ระบุ ผลลัพธ์ของการโจมตีนี้คือฐานข้อมูลส่งคืนชุดข้อมูลที่เป็นการรวมกันของผลลัพธ์ของการสืบค้นครั้งแรกดั้งเดิมและผลลัพธ์ของการสืบค้นที่สองที่แทรกเข้าไป
Example : จากตัวอย่างที่รันอยู่ ผู้โจมตีสามารถใส่ข้อความ "' UNION SELECT cardNo from CreditCards โดยที่ acctNo=10032 -- -" ลงในช่องการเข้าสู่ระบบ ซึ่งสร้างข้อความค้นหาต่อไปนี้ :

```
SELECT accounts FROM users WHERE login='' UNION
SELECT cardNo from CreditCards where
acctNo=10032 -- AND pass='' AND pin=
```

สมมติว่าไม่มีการเข้าสู่ระบบเท่ากับ "" การสืบค้นครั้งแรกดั้งเดิมจะส่งคืนชุดค่าว่าง ในขณะที่การสืบค้นที่สองส่งคืนข้อมูลจากตาราง "บัตรเครดิต" ในกรณีนี้ ฐานข้อมูลจะส่งคืนคอลัมน์ "cardNo" สำหรับบัญชี "10032" ฐานข้อมูลนำผลลัพธ์ของแบบสอบถามทั้งสองนี้ รวมเข้าด้วยกัน และส่งกลับไปยังแอปพลิเคชัน ฐานข้อมูลนำผลลัพธ์ของแบบสอบถามทั้งสองนี้ รวมเข้าด้วยกัน และส่งกลับไปยังแอปพลิเคชัน ในหลาย ๆ แอปพลิเคชัน ผลกระทบของการดำเนินการนี้คือค่า "cardNo" จะแสดงพร้อมกับข้อมูลบัญชี

Piggy-Backed Queries

Attack Intent : ดึงข้อมูล เพิ่มหรือแก้ไขข้อมูล ดำเนินการปฏิเสธบริการ ดำเนินการคำสั่งจากระยะไกล
Description : ในการโจมตีประเภทนี้ ผู้โจมตีพยายามใส่คำค้นหาเพิ่มเติมลงในแบบสอบถามเดิม เราแยกประเภทนี้ออกจากประเภทอื่นเพราะในกรณีนี้ ผู้โจมตีไม่ได้พยายามแก้ไขข้อความค้นหาเดิมที่ตั้งใจไว้ แต่พวกเขากำลังพยายามรวมข้อความค้นหาใหม่ที่แตกต่างออกไปซึ่ง "ใช้ย้อนกลับ" ในข้อความค้นหาเดิม เป็นผลให้ฐานข้อมูลได้รับแบบสอบถาม SQL หลายรายการ ข้อแรกคือการสืบค้นที่ต้องการซึ่งดำเนินการตามปกติ อันที่ตามมาคือคำถามแบบฉีด ซึ่งดำเนินการเพิ่มเติมจากครั้งแรก การโจมตีประเภทนี้อาจเป็นอันตรายอย่างยิ่ง หากสำเร็จ ผู้โจมตีสามารถแทรกคำสั่ง SQL แทบทุกประเภท รวมถึงกระบวนงานที่เก็บไว้1 ลงในข้อความค้นหาเพิ่มเติมและสั่งให้ดำเนินการไปพร้อมกับการสืบค้นข้อมูลเดิม ช่องโหว่ของการโจมตีประเภทนี้มักจะขึ้นอยู่กับการกำหนดค่าฐานข้อมูลที่อนุญาตให้มีคำสั่งหลายคำสั่งในสตริงเดียว
Example : หากผู้โจมตีป้อน "'; ผู้ใช้ตารางดรอป – –" ในช่องรหัสผ่านแอปพลิเคชันสร้างแบบสอบถาม :

```
SELECT accounts FROM users WHERE login='doe' AND
pass=''; drop table users -- ' AND pin=123
```

หลังจากเสร็จสิ้นการสอบถามครั้งแรก ฐานข้อมูลจะรู้จัก
ขั้นตอนการจัดเก็บเป็นกิจวัตรที่จัดเก็บไว้ในฐานข้อมูลและเรียกใช้โดยกลไกจัดการฐานข้อมูล ขั้นตอนเหล่านี้สามารถเป็นขั้นตอนที่ผู้ใช้กำหนดหรือขั้นตอนที่จัดทำโดยฐานข้อมูลโดยค่าเริ่มต้น ตัวคั่นข้อความค้นหา (";") และดำเนินการคำสั่งที่สองที่แทรกเข้าไป ผลลัพธ์ของการดำเนินการแบบสอบถามที่สองคือการวางตาราง users, ซึ่งน่าจะทำลายข้อมูลที่มีค่า

สมมติว่าไม่มีการเข้าสู่ระบบเท่ากับ "" การสืบค้นครั้งแรกดั้งเดิมจะส่งคืนชุดค่าว่าง ในขณะที่การสืบค้นที่สองส่งคืนข้อมูลจากตาราง "บัตรเครดิต" ในกรณีนี้ ฐานข้อมูลจะส่งคืนคอลัมน์ "cardNo" สำหรับบัญชี "10032" ฐานข้อมูลนำผลลัพธ์ของแบบสอบถามทั้งสองนี้ รวมเข้าด้วยกัน และส่งกลับไปยังแอปพลิเคชัน ฐานข้อมูลนำผลลัพธ์ของแบบสอบถามทั้งสองนี้ รวมเข้าด้วยกัน และส่งกลับไปยังแอปพลิเคชัน ในหลาย ๆ แอปพลิเคชัน ผลกระทบของการดำเนินการนี้คือค่า "cardNo" จะแสดงพร้อมกับข้อมูลบัญชี แบบสอบถามประเภทอื่นสามารถแทรกผู้ใช้ใหม่ลงในฐานข้อมูลหรือดำเนินการตามขั้นตอนที่เก็บไว้ โปรดทราบว่าฐานข้อมูลจำนวนมากไม่ต้องการอักขระพิเศษในการแยกข้อความค้นหาที่แตกต่างกัน ดังนั้นการสแกนหาตัวคั่นข้อความค้นหาจึงไม่ใช่วิธีที่มีประสิทธิภาพในการป้องกันการโจมตีประเภทนี้

Piggy-Backed Queries

Attack Intent : ดำเนินการยกระดับสิทธิ์ ดำเนินการปฏิเสธบริการ ดำเนินการคำสั่งระยะไกล

Description : SQLIA ประเภทนี้พยายามเรียกใช้โพรซีเดอร์ที่เก็บไว้ในฐานข้อมูล ทุกวันนี้ ผู้จำหน่ายฐานข้อมูลส่วนใหญ่จัดส่งฐานข้อมูลด้วยชุดโพรซีเดอร์มาตรฐานที่ขยายฟังก์ชันการทำงานของฐานข้อมูลและอนุญาตให้มีปฏิสัมพันธ์กับระบบปฏิบัติการได้ ดังนั้น เมื่อผู้โจมตีกำหนดได้ว่าฐานข้อมูลส่วนหลังใดใช้งานอยู่ สามารถสร้าง SQLIA เพื่อดำเนินการตามขั้นตอนที่จัดเก็บไว้โดยฐานข้อมูลเฉพาะนั้น ซึ่งรวมถึงขั้นตอนที่โต้ตอบกับระบบปฏิบัติการ

เป็นความเข้าใจผิดทั่วไปที่ว่าการใช้กระบวนงานที่เก็บไว้เพื่อเขียนเว็บแอปพลิเคชันทำให้พวกมันคงกระพันกับ SQLIA นักพัฒนามักแปลกใจที่พบว่าขั้นตอนการจัดเก็บของพวกเขาอาจเสี่ยงต่อการถูกโจมตีได้พอๆ กับแอปพลิเคชันปกติของพวกเขา นอกจากนี้ เนื่องจากขั้นตอนการจัดเก็บมักเขียนด้วยภาษาสคริปต์พิเศษ จึงอาจมีช่องโหว่ประเภทอื่นๆ เช่น บัฟเฟอร์ล้น ซึ่งทำให้ผู้โจมตีสามารถเรียกใช้โค้ดตามอำเภอใจบนเซิร์ฟเวอร์หรือยกระดับสิทธิ์ของตนได้

```
CREATE PROCEDURE DBO.isAuthenticated
 @userName varchar2, @pass varchar2, @pin int
AS
 EXEC("SELECT accounts FROM users
 WHERE login='" +@userName+ "' and pass='" +@password+
 "' and pin=" +@pin);
GO
```

Description :

Figure 2: Stored procedure for checking credentials

Example : ตัวอย่างนี้แสดงให้เห็นว่ากระบวนงานที่เก็บไว้แบบกำหนดพารามิเตอร์สามารถใช้ประโยชน์ผ่าน SQLIA ได้อย่างไร ในตัวอย่าง เราคิดว่าสตริงการสืบค้นที่สร้างที่บรรทัดที่ 5, 6 และ 7 ของตัวอย่างของเราถูกแทนที่ด้วยการเรียกไปยังขั้นตอนการจัดเก็บที่กำหนดไว้ในรูปที่ 2 กระบวนงานที่เก็บไว้จะส่งกลับค่าจริง/เท็จเพื่อระบุว่าข้อมูลประจำตัวของผู้ใช้ได้รับการพิสูจน์ตัวตนอย่างถูกต้องหรือไม่ ในการเปิดใช้งาน SQLIA ผู้โจมตีเพียงแค่ใส่ " ' ; ปิดตัวลง; –" ลงในช่อง userName หรือ password การ inject นี้ทำให้กระบวนงานที่เก็บไว้สร้างแบบสอบถามต่อไปนี้ :

```
SELECT accounts FROM users WHERE
login='doe' AND pass=' '; SHUTDOWN; -- AND pin=
```

ณ จุดนี้ การโจมตีนี้ทำงานเหมือนกับ piggy-back attack คิวรีแรกดำเนินการตามปกติ จากนั้นคิวรีที่สอง คิวรีที่เป็นอันตรายจะถูกดำเนินการ ซึ่งส่งผลให้ฐานข้อมูลปิดตัวลง ตัวอย่างนี้แสดงให้เห็นว่ากระบวนงานที่เก็บไว้อาจเสี่ยงต่อการโจมตีช่วงเดียวกันกับรหัสแอปพลิเคชันแบบเดิม

Inference

Attack Intent : การระบุพารามิเตอร์ที่ฉีดได้ การดึงข้อมูล การกำหนดสคีมาฐานข้อมูล

Description : ในการโจมตีนี้ แบบสอบถามจะถูกปรับเปลี่ยนเพื่อหล่อใหม่ในรูปแบบของการดำเนินการที่ดำเนินการตามคำตอบของคำถามจริง / เท็จเกี่ยวกับค่าข้อมูลในฐานข้อมูล ในการ inject ประเภทนี้ ผู้โจมตีมักจะพยายามโจมตีไซต์ที่มีความปลอดภัยเพียงพอ เพื่อที่เมื่อการฉีดสำเร็จ จะไม่มีคำติชมที่ใช้งานได้ผ่านข้อความแสดงข้อผิดพลาดของฐานข้อมูล เนื่องจากข้อความแสดงข้อผิดพลาดของฐานข้อมูลไม่สามารถให้คำติชมแก่ผู้โจมตีได้ ผู้โจมตีจึงต้องใช้วิธีการอื่นในการรับการตอบสนองจากฐานข้อมูล ในสถานการณ์นี้ ผู้โจมตีจะใส่คำสั่งเข้าไปในไซต์แล้วสังเกตว่าหน้าที่ / การตอบสนองของเว็บไซต์เปลี่ยนแปลงไปอย่างไร ด้วยการสังเกตอย่างรอบคอบเมื่อไซต์ทำงานเหมือนกันและเมื่อพฤติกรรมเปลี่ยนไป ผู้โจมตีสามารถอนุมานได้ไม่เพียงแต่ว่าพารามิเตอร์บางตัวมีความเสี่ยงเท่านั้น แต่ยังรวมถึงข้อมูลเพิ่มเติมเกี่ยวกับค่าในฐานข้อมูลด้วย มีสองเทคนิคการโจมตีที่รู้จักกันดีซึ่งอิงจากการอนุมาน อนุญาตให้ผู้โจมตีดึงข้อมูลจากฐานข้อมูลและตรวจจับพารามิเตอร์ที่มีช่องโหว่ นักวิจัยรายงานว่าด้วยเทคนิคเหล่านี้ พวกเขาสามารถบรรลุอัตราการสกัดข้อมูลที่ 1B/s

Blind Injection : ในเทคนิคนี้ ข้อมูลจะต้องอนุมานจากพฤติกรรมของเพจโดยการถามคำถามจริง/เท็จของเซิร์ฟเวอร์ หากคำสั่งที่ฉีดมีค่าเป็น จริง ไซต์จะยังคงทำงานได้ตามปกติ หากคำสั่งประเมินเป็นเท็จ แม้ว่าจะไม่มีข้อความแสดงข้อผิดพลาดที่อธิบาย หน้านั้นจะแตกต่างอย่างมากจากหน้าที่ใช้งานได้ตามปกติ

Timing Attacks : การโจมตีตามเวลาช่วยให้ผู้โจมตีได้รับข้อมูลจากฐานข้อมูลโดยสังเกตการหน่วงเวลาในการตอบสนองของฐานข้อมูล การโจมตีนี้คล้ายกับการ blind injection มาก แต่ใช้วิธีการอนุมานที่ต่างออกไป ในการดำเนินการโจมตีตามเวลา ผู้โจมตีจัดโครงสร้างการสืบค้นที่ inject เข้าไปในรูปแบบของคำสั่ง if / then ซึ่งภาคแสดงสาขาสอดคล้องกับเนื้อหาที่ไม่รู้จักเกี่ยวกับเนื้อหาของฐานข้อมูล ผู้โจมตีใช้โครงสร้าง SQL ที่ใช้เวลานานในการดำเนินการตามสาขาใดสาขาหนึ่ง (เช่น คำหลัก WAITFOR ซึ่งทำให้ฐานข้อมูลล่าช้าในการตอบสนองตามเวลาที่กำหนด) โดยการวัดการเพิ่มขึ้นหรือลดลงของเวลาตอบสนองของฐานข้อมูล ผู้โจมตีสามารถสรุปได้ว่าสาขาใดถูกนำไปใช้ในการฉีดของเขา และด้วยเหตุนี้จึงเป็นคำตอบสำหรับคำถามที่ฉีดเข้าไป

Example : การใช้โค้ดจากตัวอย่างการรันของเรา เราแสดงให้เห็นสองวิธีที่สามารถใช้การโจมตีโดยใช้การอนุมานได้ อย่างแรกคือการระบุพารามิเตอร์ที่ฉีดได้โดยใช้การ blind injection

พิจารณาการฉีดที่เป็นไปได้สองครั้งในช่องเข้าสู่ระบบ อันดับแรกคือ "legalUser" และ 1=0 – –" และอันที่สองคือ "legalUser" และ 1=1 – –" การ inject เหล่านี้ส่งผลให้เกิดคำถามสองข้อต่อไปนี้ :

SELECT accounts FROM users WHERE login='legalUser' and 1=0 -- ' AND pass='' AND pin=0
SELECT accounts FROM users WHERE login='legalUser' and 1=1 -- ' AND pass='' AND pin=0

ตอนนี้ ให้เราพิจารณาสองสถานการณ์ ในสถานการณ์แรก เรามีแอปพลิเคชันที่ปลอดภัย และอินพุตสำหรับการเข้าสู่ระบบได้รับการตรวจสอบอย่างถูกต้อง ในกรณีนี้ การ inject ทั้งสองจะส่งคืนข้อความแสดงข้อผิดพลาดในการเข้าสู่ระบบ และผู้โจมตีจะรู้ว่าพารามิเตอร์การเข้าสู่ระบบไม่มีช่องโหว่ ในสถานการณ์ที่สอง เรามีแอปพลิเคชันที่ไม่ปลอดภัยและพารามิเตอร์การเข้าสู่ระบบมีความเสี่ยงที่จะถูกแทรก ผู้โจมตีส่งการ inject ครั้งแรกและเนื่องจากจะประเมินว่าเป็นเท็จเสมอ แอปพลิเคชันจึงส่งคืนข้อความแสดงข้อผิดพลาดในการเข้าสู่ระบบ อย่างไรก็ตาม ณ จุดนี้ผู้โจมตีไม่ทราบว่าเป็นเพราะแอปพลิเคชันตรวจสอบอินพุตอย่างถูกต้องและบล็อกความพยายามในการโจมตีหรือเนื่องจากการโจมตีทำให้เกิดข้อผิดพลาดในการเข้าสู่ระบบ ผู้โจมตีจะส่งแบบสอบถามที่สอง ซึ่งจะประเมินว่าเป็นจริงเสมอ หากในกรณีนี้ไม่มีข้อความแสดงข้อผิดพลาดในการเข้าสู่ระบบ แสดงว่าผู้โจมตีรู้ว่ามีการโจมตีเกิดขึ้น และพารามิเตอร์การเข้าสู่ระบบมีความเสี่ยงที่จะถูกแทรกแซง

วิธีที่สองที่สามารถใช้การโจมตีโดยใช้การอนุมานได้คือทำการดึงข้อมูล ในที่นี้เราจะอธิบายวิธีใช้การโจมตีโดยอนุมานตามระยะเวลาเพื่อดึงชื่อตารางออกจากฐานข้อมูล ในการโจมตีนี้ สิ่งต่อไปนี้จะถูกแทรกเข้าไปในพารามิเตอร์การเข้าสู่ระบบ :

''legalUser' and ASCII(SUBSTRING((select top 1 name from sysobjects),1,1)) > X WAITFOR 5 --''.

สิ่งนี้สร้างแบบสอบถามต่อไปนี้ :

```
SELECT accounts FROM users WHERE login='legalUser' and
ASCII(SUBSTRING((select top 1 name from sysobjects),1,1))
> X WAITFOR 5 -- ' AND pass='' AND pin=0
```

ในการโจมตีนี้ ฟังก์ชัน SUBSTRING ใช้เพื่อแยกอักขระตัวแรกของชื่อตารางแรก การใช้กลยุทธ์การค้นหาแบบไบนารี ผู้โจมตีสามารถถามคำถามเกี่ยวกับตัวละครตัวนี้ได้เป็นชุด ในกรณีนี้ ผู้โจมตีจะถามว่าค่า ASCII ของอักขระนั้นมากกว่าหรือน้อยกว่าหรือเท่ากับค่าของ X หรือไม่ หากค่ามากกว่า ผู้โจมตีรู้สิ่งนี้โดยสังเกตการหน่วงเวลาเพิ่มเติม 5 วินาทีในการตอบสนองของฐานข้อมูล ผู้โจมตีสามารถใช้การค้นหาแบบไบนารีโดยเปลี่ยนค่าของ X เพื่อระบุค่าของอักขระตัวแรก

Alternate Encodings

Attack Intent : หลบเลี่ยงการตรวจจับ
Description : ในการโจมตีนี้ ข้อความที่แทรกจะถูกแก้ไขเพื่อหลีกเลี่ยงการตรวจจับโดยวิธีการเข้ารหัสเชิงป้องกันและเทคนิคการป้องกันอัตโนมัติอีกมากมาย การโจมตีประเภทนี้ใช้ร่วมกับการโจมตีอื่นๆ กล่าวอีกนัยหนึ่ง การเข้ารหัสแบบอื่นไม่ได้ให้วิธีพิเศษใดๆ ในการโจมตีแอปพลิเคชัน พวกเขาเป็นเพียงเทคนิคที่เปิดใช้งานที่ช่วยให้ผู้โจมตีสามารถหลบเลี่ยงเทคนิคการตรวจจับและป้องกันและใช้ประโยชน์จากช่องโหว่ที่อาจไม่สามารถหาประโยชน์ได้ เทคนิคการหลบเลี่ยงเหล่านี้มักมีความจำเป็น เนื่องจากแนวทางปฏิบัติในการเข้ารหัสเชิงป้องกันทั่วไปคือการสแกนหา "อักขระที่ไม่ดี" ที่รู้จัก เช่น เครื่องหมายคำพูดเดี่ยวและตัวดำเนินการความคิดเห็น

เพื่อหลีกเลี่ยงการป้องกันนี้ ผู้โจมตีได้ใช้วิธีอื่นในการเข้ารหัสสตริงการโจมตี (เช่น การใช้การเข้ารหัสอักขระฐานสิบหก, ASCII และ Unicode) เทคนิคการสแกนและตรวจจับทั่วไปไม่ได้พยายามประเมินสตริงที่เข้ารหัสพิเศษทั้งหมด ดังนั้นจึงทำให้ไม่สามารถตรวจจับการโจมตีเหล่านี้ได้ สาเหตุของปัญหาคือเลเยอร์ต่างๆ ในแอปพลิเคชันมีวิธีการจัดการการเข้ารหัสสำรองที่แตกต่างกัน แอปพลิเคชันอาจสแกนหาอักขระหลีกบางประเภทที่แสดงการเข้ารหัสสำรองในโดเมนภาษา

เลเยอร์อื่น (เช่น ฐานข้อมูล) อาจใช้อักขระหลีกต่างกัน หรือแม้แต่วิธีการเข้ารหัสที่ต่างกันโดยสิ้นเชิง ตัวอย่างเช่น ฐานข้อมูลสามารถใช้นิพจน์ char(120) เพื่อแสดงอักขระที่เข้ารหัสแบบสลับกัน "x" แต่ char(120) ไม่มีความหมายพิเศษในบริบทของภาษาของแอปพลิเคชัน การป้องกันตามรหัสที่มีประสิทธิภาพต่อการเข้ารหัสแบบอื่นนั้นยากต่อการนำไปใช้ในทางปฏิบัติ เนื่องจากนักพัฒนาจำเป็นต้องพิจารณาการเข้ารหัสที่เป็นไปได้ทั้งหมดที่อาจส่งผลต่อสตริงการสืบค้นที่กำหนดเมื่อส่งผ่านชั้นแอปพลิเคชันต่างๆ ดังนั้น ผู้โจมตีจึงประสบความสำเร็จอย่างมากในการใช้การเข้ารหัสสำรองเพื่อปกปิดสตริงการโจมตี
Example : เนื่องจากการโจมตีทุกประเภทสามารถแสดงโดยใช้การเข้ารหัสสำรอง ในที่นี้เราเพียงแค่ให้ตัวอย่างว่าการโจมตีที่เข้ารหัสทางเลือกที่ลึกลับอาจปรากฏขึ้นได้อย่างไร ในการโจมตีนี้ ข้อความต่อไปนี้จะถูกแทรกลงในช่องเข้าสู่ระบบ: "legalUser"; exec(0x73687574646f776e) – – ” แบบสอบถามผลลัพธ์ที่สร้างโดยแอปพลิเคชันคือ :

SELECT accounts FROM users WHERE login='legalUser';
exec(char(0x73687574646f776e)) -- AND pass='' AND pin=

ตัวอย่างนี้ใช้ฟังก์ชัน char() และการเข้ารหัสฐานสิบหก ASCII ฟังก์ชัน char() ใช้เป็นพารามิเตอร์ของการเข้ารหัสอักขระจำนวนเต็มหรือเลขฐานสิบหกและส่งคืนอินสแตนซ์ของอักขระนั้น กระแสของตัวเลขในส่วนที่สองของการฉีดคือการเข้ารหัสแบบเลขฐานสิบหก ASCII ของสตริง "SHUTDOWN" ดังนั้นเมื่อแบบสอบถามถูกตีความโดย ฐานข้อมูลก็จะส่งผลให้การดำเนินการตามฐานข้อมูลของคำสั่ง SHUTDOWN

5.PREVENTION OF SQLIAS

นักวิจัยได้เสนอเทคนิคที่หลากหลายเพื่อแก้ไขปัญหาของ SQL i8njection เทคนิคเหล่านี้มีตั้งแต่แนวทางปฏิบัติที่ดีที่สุดสำหรับการพัฒนาไปจนถึงเฟรมเวิร์กแบบอัตโนมัติสำหรับการตรวจจับและป้องกัน SQLIA ในส่วนนี้ เราจะทบทวนเทคนิคที่เสนอเหล่านี้และสรุปข้อดีและข้อเสียที่เกี่ยวข้องกับแต่ละเทคนิค

5.1Defensive Coding Practices

สาเหตุหลักของช่องโหว่ของ SQL injection คือการตรวจสอบอินพุตไม่เพียงพอ ดังนั้น วิธีแก้ปัญหาที่ตรงไปตรงมาในการกำจัดจุดอ่อนเหล่านี้คือการใช้แนวทางการเขียนโค้ดป้องกันที่เหมาะสม ในที่นี้ เราสรุปแนวทางปฏิบัติที่ดีที่สุดบางส่วนที่เสนอในเอกสารประกอบเพื่อป้องกันช่องโหว่ของ SQL injection

Input type checking : SQLIA สามารถทำได้โดยการ inject คำสั่งลงในพารามิเตอร์สตริงหรือตัวเลข แม้แต่การตรวจสอบอย่างง่ายของอินพุตดังกล่าวก็สามารถป้องกันการโจมตีได้มากมาย ตัวอย่างเช่น ในกรณีของ input ที่เป็นตัวเลข ผู้พัฒนาสามารถปฏิเสธการป้อนข้อมูลใดๆ ที่มีอักขระอื่นที่ไม่ใช่ digits ได้ นักพัฒนาซอฟต์แวร์จำนวนมากละเว้นการตรวจสอบประเภทนี้โดยไม่ได้ตั้งใจ เนื่องจากการป้อนข้อมูลของผู้ใช้มักจะแสดงในรูปแบบของสตริง โดยไม่คำนึงถึงเนื้อหาหรือวัตถุประสงค์ในการใช้งาน

Encoding of inputs : การ inject เข้าไปในพารามิเตอร์สตริงมักจะทำได้โดยใช้อักขระเมตาที่หลอกให้ตัวแยกวิเคราะห์ SQL ตีความ input ของผู้ใช้เป็นโทเค็น SQL การฉีดพารามิเตอร์สตริงมักจะทำได้โดยใช้อักขระเมตาที่หลอกให้ตัวแยกวิเคราะห์ SQL ตีความอินพุตของผู้ใช้เป็นโทเค็น SQL แม้ว่าจะเป็นไปได้ที่จะห้ามการใช้อักขระเมตาเหล่านี้ การทำเช่นนั้นจะจำกัดความสามารถของผู้ใช้ที่ไม่เป็นอันตรายในการระบุข้อมูลทางกฎหมายที่มีอักขระดังกล่าว ทางออกที่ดีกว่าคือการใช้ฟังก์ชันที่เข้ารหัสสตริงในลักษณะที่อักขระเมตาทั้งหมดถูกเข้ารหัสและตีความเป็นพิเศษโดยฐานข้อมูลเป็นอักขระปกติ

Positive pattern matching : นักพัฒนาควรสร้างรูทีนการตรวจสอบ input ที่ระบุอินพุตที่ดีแทนที่จะเป็นอินพุตที่ไม่ถูกต้อง วิธีการนี้โดยทั่วไปเรียกว่าการตรวจสอบความถูกต้อง ตรงข้ามกับการตรวจสอบความถูกต้องเชิงลบ ซึ่งค้นหาอินพุตสำหรับรูปแบบที่ต้องห้ามหรือโทเค็น SQL เนื่องจากนักพัฒนาซอฟต์แวร์อาจไม่สามารถจินตนาการถึงการโจมตีทุกประเภทที่สามารถใช้กับแอปพลิเคชันของตนได้ แต่ควรสามารถระบุข้อมูลทางกฎหมายทุกรูปแบบได้ การตรวจสอบเชิงบวกจึงเป็นวิธีที่ปลอดภัยกว่าในการตรวจสอบ input

Identification of all input sources : นักพัฒนาต้องตรวจสอบ input ทั้งหมดในแอปพลิเคชันของตน ตามที่เราสรุปไว้ในหัวข้อ 2.1 มีแหล่งข้อมูลที่เป็นไปได้มากมายในแอปพลิเคชัน หากใช้ในการสร้างแบบสอบถาม แหล่ง input เหล่านี้อาจเป็นช่องทางให้ผู้โจมตีแนะนำ SQLIA พูดง่ายๆ ก็คือ ต้องตรวจสอบแหล่ง input ทั้งหมด

แม้ว่าการเข้ารหัสเชิงป้องกันยังคงเป็นวิธีที่ดีที่สุดในการป้องกันช่องโหว่ของ SQL injection แอปพลิเคชันของพวกเขาก็มีปัญหาในทางปฏิบัติ การเข้ารหัสเชิงป้องกันมีแนวโน้มที่จะเกิดข้อผิดพลาดของมนุษย์และไม่ได้ใช้อย่างเข้มงวดและสมบูรณ์เหมือนเทคนิคอัตโนมัติ ในขณะที่นักพัฒนาส่วนใหญ่พยายามที่จะเขียนโค้ดอย่างปลอดภัย มันยากมากที่จะใช้แนวปฏิบัติในการเขียนโค้ดป้องกันอย่างเข้มงวดและถูกต้องกับแหล่งที่มาของข้อมูลทั้งหมด อันที่จริง ช่องโหว่ของ SQL injection จำนวนมากที่พบในแอปพลิเคชันจริงเกิดจากข้อผิดพลาดของมนุษย์ : นักพัฒนาลืมเพิ่มการตรวจสอบหรือทำการตรวจสอบ input ไม่เพียงพอ กล่าวอีกนัยหนึ่ง ในแอปพลิเคชันเหล่านี้ นักพัฒนาพยายามตรวจหาและป้องกัน SQLIA แต่ล้มเหลวในการดำเนินการดังกล่าวอย่างเพียงพอและในทุกตำแหน่งที่จำเป็น ตัวอย่างเหล่านี้ให้หลักฐานเพิ่มเติมเกี่ยวกับปัญหาที่เกี่ยวข้องกับการใช้การเข้ารหัสป้องกันของนักพัฒนา

นอกจากนี้ แนวทางที่อิงจากการเข้ารหัสเชิงป้องกันยังอ่อนแอลงด้วยการส่งเสริม และ การยอมรับอย่างแพร่หลายที่เรียกว่า "วิธีหลอก" เราหารือเกี่ยวกับวิธีแก้ไขหลอกที่เสนอบ่อยที่สุดสองวิธี การแก้ไขครั้งแรกประกอบด้วยการตรวจสอบการป้อนข้อมูลของผู้ใช้สำหรับคำหลักของ SQL เช่น "จาก" "ที่ไหน" และ "เลือก" และตัวดำเนินการ SQL เช่นตัวดำเนินการเครื่องหมายคำพูดหรือความคิดเห็นเดียว เหตุผลที่อยู่เบื้องหลังข้อเสนอแนะนี้คือการมีคีย์เวิร์ด และ ตัวดำเนินการดังกล่าวอาจบ่งบอกถึงความพยายามของ SQLIA วิธีการนี้ส่งผลให้ผลบวกลวงในอัตราสูงอย่างชัดเจน เนื่องจากในหลายแอปพลิเคชัน คีย์เวิร์ด SQL สามารถเป็นส่วนหนึ่งของการป้อนข้อความปกติ และตัวดำเนินการ SQL สามารถใช้เพื่อแสดงสูตรหรือแม้แต่ชื่อ (เช่น O'Brian) วิธีแก้หลอกที่แนะนำโดยทั่วไปข้อที่สองคือการใช้กระบวนงานที่เก็บไว้หรือคำสั่งที่เตรียมไว้เพื่อป้องกัน SQLIAs น่าเสียดายที่ขั้นตอนการจัดเก็บ และ คำสั่งที่เตรียมไว้อาจมีความเสี่ยงต่อ SQLIAs เว้นแต่นักพัฒนาจะใช้แนวทางการเข้ารหัสเชิงป้องกันอย่างเคร่งครัด ผู้อ่านที่สนใจอาจอ้างอิงถึงตัวอย่างของวิธีการแก้ไขหลอกเหล่านี้สามารถล้มล้างได้

5.2Detection and Prevention Techniques

นักวิจัยได้เสนอเทคนิคต่างๆ เพื่อช่วยนักพัฒนา และ ชดเชยข้อบกพร่องในการประยุกต์ใช้การเข้ารหัสเชิงป้องกัน

BlackBoxTesting Huang และ เพื่อนร่วมงานเสนอ WAVES ซึ่งเป็นเทคนิค black-box สำหรับทดสอบเว็บแอปพลิเคชันสำหรับช่องโหว่ของ SQL injection เทคนิคนี้ใช้โปรแกรมรวบรวมข้อมูลเว็บเพื่อระบุจุดทั้งหมดในเว็บแอปพลิเคชันที่สามารถใช้ในการ inject SQLIAs จากนั้นจะสร้างการโจมตีที่กำหนดเป้าหมายไปยังจุดดังกล่าวตามรายการรูปแบบและเทคนิคการโจมตีที่ระบุ จากนั้น WAVES จะตรวจสอบการตอบสนองของแอปพลิเคชันต่อการโจมตีและใช้เทคนิคการเรียนรู้ของเครื่องเพื่อปรับปรุงวิธีการโจมตี เทคนิคนี้ช่วยปรับปรุงเทคนิคการทดสอบการเจาะระบบส่วนใหญ่โดยใช้แนวทางการเรียนรู้ของเครื่องเพื่อเป็นแนวทางในการทดสอบ อย่างไรก็ตาม เช่นเดียวกับ black-box และเทคนิคการทดสอบการเจาะทั้งหมด มันไม่สามารถรับประกันความสมบูรณ์ได้

Static Code Checkers JDBC-Checker เป็นเทคนิคสำหรับการตรวจสอบความถูกต้องของประเภทการสืบค้น SQL ที่สร้างขึ้นแบบไดนามิก เทคนิคนี้ไม่ได้พัฒนาขึ้นโดยมีจุดประสงค์ในการตรวจจับ และ ป้องกัน SQLIAs ทั่วไป แต่สามารถใช้เพื่อป้องกันการโจมตีที่ใช้ประโยชน์จากประเภทที่ไม่ตรงกันในสตริงการสืบค้นที่สร้างแบบไดนามิก JDBC-Checker สามารถตรวจพบหนึ่งในสาเหตุหลักของช่องโหว่ SQLIAs ในโค้ด นั่นคือ การตรวจสอบประเภทอินพุตที่ไม่เหมาะสม อย่างไรก็ตาม เทคนิคนี้จะไม่จับรูปแบบทั่วไปของ SQLIAs เนื่องจากการโจมตีเหล่านี้ส่วนใหญ่ประกอบด้วยการสืบค้นทางวากยสัมพันธ์และพิมพ์ข้อความค้นหาที่ถูกต้อง

Wassermann และ Su เสนอแนวทางที่ใช้การวิเคราะห์แบบสถิตร่วมกับการให้เหตุผลอัตโนมัติเพื่อตรวจสอบว่าการสืบค้น SQL ที่สร้างขึ้นในเลเยอร์แอปพลิเคชันไม่สามารถมีการพูดซ้ำซาก ข้อเสียเปรียบหลักของเทคนิคนี้คือขอบเขตจำกัดเฉพาะการตรวจจับและป้องกัน tautology และ ไม่สามารถตรวจจับการโจมตีประเภทอื่นได้

Combined Static and Dynamic Analysis AMNESIA เป็นเทคนิคตามแบบจำลองที่รวมการวิเคราะห์แบบสถิต และ การตรวจสอบรันไทม์ ในระยะคงที่ AMNESIA ใช้การวิเคราะห์แบบสถิตเพื่อสร้างแบบจำลองการสืบค้นประเภทต่างๆ ที่แอปพลิเคชันสามารถสร้างได้อย่างถูกกฎหมาย ณ จุดเข้าถึงฐานข้อมูลแต่ละจุด ในช่วงไดนามิก AMNESIA จะสกัดกั้นการสืบค้นทั้งหมดก่อนที่จะถูกส่งไปยังฐานข้อมูล และ ตรวจสอบการสืบค้นแต่ละรายการเทียบกับแบบจำลองที่สร้างแบบสแตติก

แบบสอบถามที่ละเมิดโมเดลจะถูกระบุเป็น SQLIAs และ ป้องกันไม่ให้ดำเนินการบนฐานข้อมูล ในการประเมิน ผู้เขียนได้แสดงให้เห็นว่าเทคนิคนี้ทำงานได้ดีกับ SQLIAs ข้อจำกัดหลักของเทคนิคนี้คือความสำเร็จนั้นขึ้นอยู่กับความถูกต้องของการวิเคราะห์แบบคงที่สำหรับการสร้างแบบจำลองการสืบค้น โค้ดที่สร้างความสับสนหรือเทคนิคการพัฒนาคิวรีบางประเภทอาจทำให้ขั้นตอนนี้แม่นยำน้อยลงและส่งผลให้เกิดทั้งผลบวกลวง และ ผลลบลวง ในทำนองเดียวกัน แนวทางที่เกี่ยวข้องสองวิธีล่าสุด SQLGuard และ SQLCheck ยังตรวจสอบการสืบค้นที่รันไทม์เพื่อดูว่าสอดคล้องกับรูปแบบการสืบค้นที่คาดไว้หรือไม่ ในแนวทางเหล่านี้ โมเดลจะแสดงเป็นไวยากรณ์ที่ยอมรับเฉพาะการสืบค้นทางกฎหมายเท่านั้น ใน SQLGuard โมเดลจะถูกอนุมานตอนรันไทม์โดยตรวจสอบโครงสร้างของแบบสอบถามก่อน และ หลังการเพิ่ม input ของผู้ใช้ ใน SQLCheck โมเดลนั้นถูกกำหนดโดยผู้พัฒนาอย่างอิสระ ทั้งสองวิธีใช้รหัสลับเพื่อกำหนดเขตการป้อนข้อมูลของผู้ใช้ในระหว่างการแยกวิเคราะห์โดยตัวตรวจสอบรันไทม์ ดังนั้นความปลอดภัยของวิธีการจึงขึ้นอยู่กับผู้โจมตีที่ไม่สามารถค้นพบคีย์ได้ นอกจากนี้ การใช้สองแนวทางนี้กำหนดให้นักพัฒนาต้องเขียนโค้ดใหม่เพื่อใช้ไลบรารีระดับกลางพิเศษ หรือ แทรกเครื่องหมายพิเศษด้วยตนเองลงในโค้ดที่ผู้ใช้ป้อนข้อมูลถูกเพิ่มลงในแบบสอบถามที่สร้างขึ้นแบบไดนามิก

TaintBasedApproaches WebSSARI ตรวจจับข้อผิดพลาดที่เกี่ยวข้องกับการตรวจสอบอินพุตโดยใช้ข้อมูล flow การวิเคราะห์ ในแนวทางนี้ การวิเคราะห์แบบสถิตถูกใช้เพื่อตรวจสอบความสกปรกกับเงื่อนไขเบื้องต้นสำหรับฟังก์ชันที่ละเอียดอ่อน การวิเคราะห์จะตรวจจับจุดที่ยังไม่เป็นไปตามเงื่อนไขเบื้องต้น และสามารถแนะนำตัวกรองและฟังก์ชันการฆ่าเชื้อที่สามารถเพิ่มลงในแอปพลิเคชันโดยอัตโนมัติเพื่อให้เป็นไปตามเงื่อนไขเบื้องต้นเหล่านี้ ระบบ WebSSARI ทำงานโดยพิจารณาว่าเป็น input ที่ผ่านการฆ่าเชื้อแล้วซึ่งผ่านชุดตัวกรองที่กำหนดไว้ล่วงหน้า ในการประเมิน ผู้เขียนสามารถตรวจพบช่องโหว่ด้านความปลอดภัยในแอพพลิเคชั่นที่มีอยู่มากมาย ข้อเสียเปรียบหลักของเทคนิคนี้คือ สันนิษฐานว่าเงื่อนไขเบื้องต้นที่เพียงพอสำหรับฟังก์ชันที่ละเอียดอ่อนสามารถแสดงออกได้อย่างแม่นยำโดยใช้ระบบการพิมพ์ และ การป้อนข้อมูลผ่านตัวกรองบางประเภทก็เพียงพอที่จะพิจารณาว่าไม่มีมลทิน สำหรับฟังก์ชัน และ การใช้งานหลายประเภท ข้อสันนิษฐานนี้แรงเกินไป

Livshits และ Lam ใช้เทคนิคการวิเคราะห์แบบคงที่เพื่อตรวจหาช่องโหว่ในซอฟต์แวร์ วิธีการพื้นฐานคือการใช้เทคนิค flow ข้อมูลเพื่อตรวจจับเมื่อมีการใช้อินพุตที่เสียไปเพื่อสร้าง SQL query ข้อความค้นหาเหล่านี้จะถูกติดแท็กเป็นช่องโหว่ของ SQLIA ผู้เขียนแสดงให้เห็นถึงความเป็นไปได้ของเทคนิคโดยใช้วิธีการนี้เพื่อค้นหาช่องโหว่ด้านความปลอดภัยในชุดมาตรฐาน ข้อจำกัดหลักของวิธีนี้คือสามารถตรวจจับได้เฉพาะรูปแบบที่รู้จักของ SQLIAs และ เนื่องจากใช้การวิเคราะห์แบบอนุรักษ์นิยมและมีการสนับสนุนที่จำกัดสำหรับการดำเนินการที่ไม่มีมลทิน จึงสามารถสร้างผลบวกปลอมได้ค่อนข้างสูง

มีการเสนอวิธีการวิเคราะห์มลทินแบบไดนามิกหลายวิธี สองแนวทางที่คล้ายกันโดย Nguyen-Tuong และ เพื่อนร่วมงาน และ Pietraszek และ Berghe ปรับเปลี่ยนตัวแปล PHP เพื่อติดตามข้อมูลมลทินต่ออักขระอย่างแม่นยำ เทคนิคนี้ใช้การวิเคราะห์ที่มีความละเอียดอ่อนตามบริบทเพื่อตรวจหาและปฏิเสธการสืบค้น หากมีการใช้อินพุตที่ไม่น่าเชื่อถือเพื่อสร้างโทเค็น SQL บางประเภท ข้อเสียทั่วไปของสองวิธีนี้คือต้องมีการปรับเปลี่ยนสภาพแวดล้อมรันไทม์ ซึ่งส่งผลต่อการพกพา เทคนิคโดย Haldar และ เพื่อนร่วมงาน และ SecuriFly ใช้แนวทางที่คล้ายกันสำหรับ Java อย่างไรก็ตาม เทคนิคเหล่านี้ไม่ได้ใช้การวิเคราะห์ที่มีความละเอียดอ่อนตามบริบทที่ใช้โดยอีกสองวิธี และ ติดตามข้อมูลมลทินบนพื้นฐานต่อสตริง (ตรงข้ามกับอักขระแต่ละตัว) SecuriFly ยังพยายามล้างสตริงการสืบค้นที่สร้างขึ้นโดยใช้ input ที่ไม่บริสุทธิ์ อย่างไรก็ตาม วิธีการฆ่าเชื้อนี้ไม่ได้ช่วยอะไรหากฉีดเข้าไปในช่องตัวเลข โดยทั่วไปแล้ว เทคนิคที่มีเทนต์แบบไดนามิกได้แสดงให้เห็นถึงความสามารถในการตรวจจับ และ ป้องกัน SQLIAs ได้เป็นอย่างดี ข้อเสียเปรียบหลักของแนวทางเหล่านี้คือการระบุแหล่งที่มาของการป้อนข้อมูลของผู้ใช้ที่เสียไปในแอปพลิเคชันเว็บที่มีโมดูลสูง และ เผยแพร่ข้อมูลมลทินอย่างถูกต้องมักเป็นงานที่ยาก

NewQueryDevelopmentParadigms สองแนวทางล่าสุดคือ SQL DOM และ Safe Query Objects ใช้การสืบค้นฐานข้อมูลเพื่อจัดเตรียมวิธีการเข้าถึงฐานข้อมูลที่ปลอดภัย และ เชื่อถือได้ เทคนิคเหล่านี้เสนอวิธีที่มีประสิทธิภาพในการหลีกเลี่ยงปัญหา SQLIA โดยการเปลี่ยนกระบวนการสร้างแบบสอบถามจากกระบวนการที่ไม่มีการควบคุมซึ่งใช้การต่อสตริงเป็นระบบที่ใช้ API ที่ตรวจสอบประเภท ภายใน API ของพวกเขา พวกเขาสามารถใช้แนวทางปฏิบัติที่ดีที่สุดในการเข้ารหัสอย่างเป็นระบบ เช่น การกรอง input และ การตรวจสอบประเภท input ของผู้ใช้อย่างเข้มงวด

โดยการเปลี่ยนกระบวนทัศน์การพัฒนาที่สร้าง SQL queries เทคนิคเหล่านี้จะขจัดแนวปฏิบัติในการเข้ารหัสที่ทำให้ SQLIAs เป็นไปได้มากที่สุด แม้ว่าจะมีประสิทธิภาพ แต่เทคนิคเหล่านี้ก็มีข้อเสียคือต้องการให้นักพัฒนาเรียนรู้และใช้กระบวนทัศน์การเขียนโปรแกรมใหม่หรือกระบวนการพัฒนาแบบสอบถาม นอกจากนี้ เนื่องจากพวกเขามุ่งเน้นที่การใช้กระบวนการพัฒนาใหม่ จึงไม่ได้ให้การป้องกันใดๆ หรือความปลอดภัยที่ได้รับการปรับปรุงสำหรับระบบเดิมที่มีอยู่

IntrusionDetectionSystems Valeur และเพื่อนร่วมงานเสนอให้ใช้ระบบตรวจจับการบุกรุก (IDS) เพื่อตรวจจับ SQLIAs ระบบ IDS ของพวกเขาใช้เทคนิคแมชชีนเลิร์นนิงที่ได้รับการฝึกฝนโดยใช้ชุดคำถามทั่วไปของแอปพลิเคชัน เทคนิคนี้สร้างแบบจำลองของการสืบค้นข้อมูลทั่วไป แล้วตรวจสอบแอปพลิเคชันขณะรันไทม์เพื่อระบุการสืบค้นที่ไม่ตรงกับแบบจำลอง ในการประเมิน Valeur และเพื่อนร่วมงานได้แสดงให้เห็นว่าระบบของพวกเขาสามารถตรวจจับการโจมตีด้วยอัตราความสำเร็จที่สูง อย่างไรก็ตาม ข้อจำกัดพื้นฐานของเทคนิคการเรียนรู้คือไม่สามารถรับประกันความสามารถในการตรวจจับได้ เนื่องจากความสำเร็จขึ้นอยู่กับคุณภาพของชุดการฝึกที่ใช้ ชุดฝึกอบรมที่ไม่ดีจะทำให้เทคนิคการเรียนรู้สร้างผลบวกและค่าลบปลอมจำนวนมาก

Proxy Filters Security Gateway เป็นระบบการกรองพร็อกซี่ที่บังคับใช้กฎการตรวจสอบ input บนข้อมูลที่เชื่อมต่อกับเว็บแอปพลิเคชัน การใช้ภาษาอธิบายนโยบายความปลอดภัย (SPDL) นักพัฒนาจะจัดเตรียมข้อจำกัดและระบุการแปลงเพื่อใช้กับพารามิเตอร์ของแอปพลิเคชันเมื่อดำเนินการจากเว็บเพจไปยังเซิร์ฟเวอร์แอปพลิเคชัน เนื่องจาก SPDL มีความชัดเจนอย่างมาก จึงช่วยให้นักพัฒนามีอิสระอย่างมากในการแสดงนโยบายของตน อย่างไรก็ตาม วิธีการนี้เป็นแนวทางของมนุษย์ และ เช่นเดียวกับการเขียนโปรแกรมเชิงป้องกัน นักพัฒนาต้องรู้ว่าไม่เพียงแต่ข้อมูลใดที่ต้องถูกกรอง แต่ยังรวมถึงรูปแบบและตัวกรองใดบ้างที่จะนำไปใช้กับข้อมูล

Instruction Set Randomization SQLrand เป็นแนวทางที่อิงจากการสุ่มชุดคำสั่ง SQLrand มีกรอบงานที่ช่วยให้นักพัฒนาสามารถสร้างแบบสอบถามโดยใช้คำสั่งแบบสุ่มแทนคีย์เวิร์ด SQL ปกติ ตัวกรองพร็อกซีจะสกัดกั้นการสืบค้นไปยังฐานข้อมูล และ ยกเลิกการสุ่มคำหลัก โค้ด SQL ที่ผู้โจมตีใส่เข้าไปจะไม่ถูกสร้างขึ้นโดยใช้ชุดคำสั่งแบบสุ่ม ดังนั้น คำสั่งที่ฉีดเข้าไปจะส่งผลให้เกิดการสืบค้นทางไวยากรณ์ที่ไม่ถูกต้อง

แม้ว่าเทคนิคนี้จะมีประสิทธิภาพมาก แต่ก็มีข้อเสียในทางปฏิบัติหลายประการ ประการแรก เนื่องจากใช้รหัสลับเพื่อแก้ไขคำสั่ง ความปลอดภัยของวิธีการจึงขึ้นอยู่กับผู้โจมตีที่ไม่สามารถค้นพบคีย์ได้ ประการที่สอง วิธีการกำหนดค่าใช้จ่ายโครงสร้างพื้นฐานที่สำคัญเนื่องจากต้องมีการรวมพร็อกซีสำหรับฐานข้อมูลในระบบ

Technique	Taut	Illegal / Incorrect	Piggy-back	Union	Stored Proc.	Infer.	Alt. Encodings.
AMNESIA	●	●	●	●	×	●	●
CSS	●	●	●	●	×	●	×
IDS	○	○	○	○	○	○	○
Java Dynamic Tainting	—	—	—	—	—	—	—
SQLCheck	●	●	●	●	×	●	●
SQLGuard	●	●	●	●	×	●	●
SQLrand	●	×	●	●	×	●	×
Tautology-checker	●	×	×	×	×	×	×
Web App. Hardening	●	●	●	●	×	●	×

Table 1: Comparison of detection-focused techniques with respect to attack types

Technique	Taut	Illegal / Incorrect	Piggy-back	Union	Stored Proc.	Infer.	Alt. Encodings.
JDBC-Checker	—	—	—	—	—	—	—
Java Static Tainting*	●	●	●	●	●	●	●
Safe Query Objects	●	●	●	●	×	●	●
Security Gateway*	—	—	—	—	—	—	—
SecuriFly	—	—	—	—	—	—	—
SQL DOM	●	●	●	●	×	●	●
WAVES	○	○	○	○	○	—	○
WebSSARI*	●	●	●	●	●	●	●

Table 2: Comparison of prevention-focused techniques with respect to attack types

6.TECHNIQUES EVALUATION

ในส่วนนี้ เราประเมินเทคนิคที่นำเสนอในส่วนที่ 5 โดยใช้เกณฑ์ต่างๆ ที่หลากหลาย ขั้นแรกเราจะพิจารณาว่าการโจมตีประเภทใดที่แต่ละเทคนิคสามารถจัดการได้ สำหรับชุดย่อยของเทคนิคที่ยึดตามการปรับปรุงโค้ด เราจะพิจารณาว่าแนวปฏิบัติด้านการเขียนโค้ดป้องกันแบบใดที่เทคนิคนี้ช่วยบังคับใช้ จากนั้นเราจะระบุกลไกการฉีดที่แต่ละเทคนิคสามารถจัดการได้ สุดท้าย เราประเมินข้อกำหนดการปรับใช้ของแต่ละเทคนิค

6.1Evaluation with Respect to Attack Types

เราประเมินแต่ละเทคนิคที่เสนอเพื่อประเมินว่าสามารถจัดการกับการโจมตีประเภทต่างๆ ที่แสดงในส่วนที่ 4 ได้ หรือ ไม่ สำหรับเทคนิคที่พิจารณาส่วนใหญ่แล้ว เราไม่สามารถเข้าถึงการนำไปใช้งานได้ เนื่องจากเทคนิคดังกล่าวไม่ได้ถูกนำไปใช้หรือไม่มีการนำไปใช้ ดังนั้นเราจึงประเมินเทคนิคในเชิงวิเคราะห์ ซึ่งต่างจากการประเมินกับการโจมตีจริง สำหรับเทคนิคที่อิงกับนักพัฒนา นั่นคือเทคนิคที่ต้องการการแทรกแซงของนักพัฒนา เราคิดว่านักพัฒนาสามารถใช้แนวทางปฏิบัติด้านการเข้ารหัสที่จำเป็นทั้งหมดได้อย่างถูกต้อง กล่าวอีกนัยหนึ่งการประเมินเทคนิคเหล่านี้มองในแง่ดีเมื่อเปรียบเทียบกับประสิทธิภาพในทางปฏิบัติ ในตารางของเรา เราแสดงถึงเทคนิคที่อิงกับนักพัฒนาโดยใช้สัญลักษณ์ “*”

เพื่อวัตถุประสงค์ในการเปรียบเทียบ เราแบ่งเทคนิคออกเป็นสองกลุ่ม : เทคนิคที่เน้นการป้องกัน และ เทคนิคที่เน้นการตรวจจับ เทคนิคที่เน้นการป้องกันเป็นเทคนิคที่ระบุช่องโหว่ในโค้ดแบบคงที่ เสนอกระบวนทัศน์การพัฒนาที่แตกต่างกันสำหรับแอปพลิเคชันที่สร้างการสืบค้น SQL หรือ เพิ่มการตรวจสอบในแอปพลิเคชันเพื่อบังคับใช้แนวทางปฏิบัติที่ดีที่สุดในการเข้ารหัสการป้องกัน (ดูหัวข้อ 5.1) เทคนิคที่เน้นการตรวจจับเป็นเทคนิคที่ตรวจจับการโจมตีส่วนใหญ่ขณะรันไทม์

Tables 1 และ 2 สรุปผลการประเมินของเรา เราใช้เครื่องหมายสี่ประเภทที่แตกต่างกันเพื่อระบุว่าเทคนิคดำเนินการอย่างไรโดยคำนึงถึงประเภทการโจมตีที่กำหนด เราใช้สัญลักษณ์ “•” เพื่อแสดงว่าเทคนิคหนึ่งสามารถหยุดการโจมตีประเภทนั้นได้สำเร็จ ในทางกลับกัน เราใช้สัญลักษณ์ “×” เพื่อแสดงว่าเทคนิคไม่สามารถหยุดการโจมตีประเภทนั้นได้ เราใช้สัญลักษณ์ที่แตกต่างกันสองแบบเพื่อจำแนกเทคนิคที่มีประสิทธิภาพเพียงบางส่วนเท่านั้น สัญลักษณ์ “ ∘ ” หมายถึงเทคนิคที่สามารถระบุประเภทการโจมตีที่พิจารณา แต่ไม่สามารถรับประกันความสมบูรณ์ได้

ตัวอย่างของเทคนิคดังกล่าวจะเป็นเทคนิคการทดสอบกล่องดำ เช่น WAVES หรือแนวทาง IDS จาก Valeur และ เพื่อนร่วมงาน สัญลักษณ์ "-" หมายถึง เทคนิคที่กล่าวถึงประเภทการโจมตีที่พิจารณาเพียงบางส่วนเท่านั้นเนื่องจากข้อจำกัดที่แท้จริงของแนวทางที่อยู่เบื้องหลัง ตัวอย่างเช่น JDBCChecker ตรวจพบข้อผิดพลาดเกี่ยวกับประเภทที่เปิดใช้งานช่องโหว่ของ SQL injection อย่างไรก็ตาม เนื่องจากข้อผิดพลาดเกี่ยวกับประเภทเป็นเพียงหนึ่งในสาเหตุที่เป็นไปได้หลายประการของช่องโหว่ของ SQL injection แนวทางนี้จึงถูกจัดประเภทว่าจัดการการโจมตีแต่ละประเภทเพียงบางส่วนเท่านั้น

ครึ่งหนึ่งของเทคนิคที่เน้นการป้องกันสามารถจัดการกับการโจมตีทุกประเภทได้อย่างมีประสิทธิภาพ เทคนิคบางอย่างมีผลเพียงบางส่วนเท่านั้น: JDBC-Checker โดยคำจำกัดความระบุเฉพาะชุดย่อยของ SQLIAs; Security Gateway เนื่องจากไม่สามารถจัดการแหล่งที่มาของการฉีดได้ทั้งหมด (ดูหัวข้อ 6.2) ไม่สามารถระบุโปรไฟล์การโจมตีทั้งหมดได้อย่างสมบูรณ์ SecuriFly เนื่องจากวิธีการป้องกันคือการหลีกเลี่ยงอักขระเมตาของ SQL ทั้งหมด ซึ่งยังคงอนุญาตให้ฉีดเข้าไปในช่องตัวเลข และ WAVES ซึ่งเนื่องจากเป็นเทคนิคการทดสอบจึงไม่สามารถรับประกันความสมบูรณ์ได้ เราเชื่อว่าโดยรวมแล้ว เทคนิคที่เน้นการป้องกันนั้นทำได้ดีเพราะรวมเอาแนวปฏิบัติด้านการเข้ารหัสเพื่อการป้องกันเข้าไว้ในกลไกการป้องกันด้วย ดูหัวข้อ 6.4 สำหรับการอภิปรายเพิ่มเติมในหัวข้อนี้

เทคนิคที่เน้นการตรวจจับส่วนใหญ่ทำงานได้ดีกับการโจมตีประเภทต่างๆ ข้อยกเว้นสามประการคือแนวทาง IDS ตามแนวทางของ Valeur และเพื่อนร่วมงาน ซึ่งประสิทธิภาพขึ้นอยู่กับคุณภาพของชุดการฝึกที่ใช้ Java Dynamic Tainting ซึ่งประสิทธิภาพจะได้รับผลกระทบในทางลบจากข้อเท็จจริงที่ว่าการดำเนินการที่ไม่มีมลทินนั้นอนุญาตให้ใช้อินพุตโดยไม่คำนึงถึงคุณภาพ ของการตรวจสอบ และ tautology-checker ซึ่งโดยคำจำกัดความสามารถระบุได้เฉพาะการโจมตีแบบ tautology เท่านั้น

การโจมตีสองประเภท กระบวนงานที่เก็บไว้และการเข้ารหัสสำรอง ทำให้เกิดปัญหากับเทคนิคส่วนใหญ่ ด้วยกระบวนงานที่เก็บไว้ รหัสที่สร้างแบบสอบถามจะถูกจัดเก็บและดำเนินการบนฐานข้อมูล

Technique	Modify Code Base	Detection	Prevention	Additional
AMNESIA	No	Automated	Automated	None
CSSE	No	Automated	Automated	Custom PHP Interpreter
IDS	No	Automated	Generate Report	IDS System-Training Set
JDBC-Checker	No	Automated	Code Suggestions	None
Java Dynamic Tainting	No	Automated	Automated	None
Java Static Tainting	No	Automated	Code Suggestions	None
Safe Query Objects	Yes	N/A	Automated	Developer Training
SecuriFly	No	Automated	Automated	None
Security Gateway	No	Manual Specification	Automated	Proxy Filter
SQLCheck	Yes	Semi-Automated	Automated	Key Management
SQLGuard	Yes	Semi-Automated	Automated	None
SQL DOM	Yes	N/A	Automated	Developer Training
SQLrand	Yes	Automated	Automated	Proxy, Developer Training, Key Management
Tautology-checker	No	Automated	Code Suggestions	None
WAVES	No	Automated	Generate Report	None
Web App. Hardening	No	Automated	Automated	Custom PHP Interpreter
WebSSARI	No	Automated	Semi-Automated	None

Table 3: Comparison of techniques with respect to deployment requirements

Technique	Input type checking	Encoding of input	Identification of all input sources	Positive pattern matching
JDBC-Checker	Yes	No	No	No
Java Static Tainting	No	No	Yes	No
Safe Query Objects	Yes	Yes	N/A	No
SecuriFly	No	Yes	Yes	No
Security Gateway	Yes	Yes	No	Yes
SQL DOM	Yes	Yes	N/A	No
WebSSARI	Yes	Yes	Yes	Yes

Table 4: Evaluation of Code Improvement Techniques with Respect to Common Development Errors

เทคนิคส่วนใหญ่ที่พิจารณาจะเน้นเฉพาะการสืบค้นที่สร้างขึ้นภายในแอปพลิเคชันเท่านั้น การขยายเทคนิคเพื่อให้ครอบคลุมการสืบค้นที่สร้าง และ ดำเนินการบนฐานข้อมูลนั้นไม่ตรงไปตรงมา และ โดยทั่วไปต้องใช้ความพยายามอย่างมาก ด้วยเหตุนี้ การโจมตีตามขั้นตอนการจัดเก็บจึงเป็นปัญหาสำหรับเทคนิคต่างๆ การโจมตีโดยใช้การเข้ารหัสสำรองก็ยากต่อการจัดการเช่นกัน มีเพียงสามเทคนิคเท่านั้น AMNESIA, SQLCheck และ SQLGuard ที่จัดการกับการโจมตีประเภทนี้อย่างชัดเจน เหตุผลที่เทคนิคเหล่านี้ประสบความสำเร็จในการโจมตีดังกล่าว ก็คือพวกเขาใช้ฐานข้อมูล lexer หรือ parser เพื่อตีความสตริงการสืบค้นในลักษณะเดียวกับที่ฐานข้อมูลใช้ เทคนิคอื่นๆ ที่ทำคะแนนได้ดีในหมวดหมู่นี้คือเทคนิคที่อิงกับนักพัฒนา (เช่น Java Static Tainting และ WebSSARI) หรือเทคนิคที่แก้ไขปัญหาโดยใช้ API มาตรฐาน (เช่น SQL DOM และ Safe Query Objects)

สิ่งสำคัญคือต้องสังเกตว่าเราไม่ได้คำนึงถึงความแม่นยำในการประเมินของเรา เทคนิคหลายอย่างที่เราพิจารณาอยู่บนพื้นฐานของการวิเคราะห์เชิงอนุรักษ์นิยม หรือ สมมติฐานที่อาจส่งผลให้เกิดผลบวกที่ผิดพลาด อย่างไรก็ตาม เนื่องจากเราไม่มีวิธีที่ถูกต้องในการจำแนกความถูกต้องของเทคนิคดังกล่าว ขาดการดำเนินการทั้งหมด และ ประเมินผลการปฏิบัติงานของข้อมูลที่ถูกต้องตามกฎหมายจำนวนมาก เราจึงไม่ได้พิจารณาคุณลักษณะนี้ในการประเมินของเรา

6.2EvaluationwithRespecttoInjectionMechanisms

เราประเมินแต่ละเทคนิคเกี่ยวกับการจัดการกลไกการ inject ต่างๆ ที่เรากำหนดไว้ในส่วนที่ 2.1 แม้ว่าเทคนิคส่วนใหญ่ไม่ได้ระบุถึงกลไกการฉีดทั้งหมดอย่างเฉพาะเจาะจง แต่ทั้งสองวิธีสามารถขยายออกได้อย่างง่ายดายเพื่อจัดการกับกลไกดังกล่าวทั้งหมด ยกเว้นสองวิธี ข้อยกเว้นสองประการคือ Security Gateway และ WAVES Security Gateway สามารถตรวจสอบได้เฉพาะพารามิเตอร์ URL และ ฟิลด์คุกกี้ เนื่องจากอยู่บนเครือข่ายระหว่างแอปพลิเคชัน และ ผู้โจมตีจึงไม่สามารถตรวจสอบตัวแปรเซิร์ฟเวอร์ และ แหล่งที่มาของการฉีดลำดับที่สองซึ่งไม่ผ่านเกตเวย์ WAVES สามารถระบุการแทรกผ่านการป้อนข้อมูลของผู้ใช้เท่านั้น เนื่องจากจะสร้างการโจมตีที่ส่งไปยังแอปพลิเคชันผ่านแบบฟอร์มหน้าเว็บเท่านั้น

6.3Evaluation with Respect to Deployment Requirements

แต่ละเทคนิคมีข้อกำหนดในการปรับใช้ที่แตกต่างกัน ในการพิจารณาความพยายาม และ โครงสร้างพื้นฐานที่จำเป็นในการใช้เทคนิค เราได้ตรวจสอบคำอธิบายของผู้เขียนเกี่ยวกับเทคนิค และ การใช้งานในปัจจุบัน เราประเมินแต่ละเทคนิคตามเกณฑ์ต่อไปนี้ : (1) เทคนิคนี้ต้องการให้นักพัฒนาแก้ไขฐานโค้ดหรือไม่ (2) ระดับของระบบอัตโนมัติในด้านการตรวจจับของแนวทางคืออะไร? (3) ระดับของระบบอัตโนมัติในด้านการป้องกันของแนวทางคืออะไร? (4) โครงสร้างพื้นฐานใด (ไม่รวมเครื่องมือเอง) ที่จำเป็นในการใช้เทคนิคให้สำเร็จ ผลลัพธ์ของการจำแนกประเภทนี้สรุปไว้ในตารางที่ 3

6.4Evaluation of Prevention-Focused Techniques with Respect to Defensive Coding Practices

การประเมินเทคนิคเบื้องต้นของเรากับการโจมตีประเภทต่างๆ บ่งชี้ว่าเทคนิคที่เน้นการป้องกันนั้นทำงานได้ดีมากกับการโจมตีเหล่านี้ส่วนใหญ่ เราตั้งสมมติฐานว่าผลลัพธ์นี้เกิดจากข้อเท็จจริงที่ว่าเทคนิคการป้องกันหลายอย่างกำลังใช้แนวทางปฏิบัติที่ดีที่สุดในการเขียนโค้ดป้องกันกับฐานโค้ด ดังนั้นเราจึงตรวจสอบแต่ละเทคนิคที่เน้นการป้องกันและจัดประเภทตามแนวทางปฏิบัติการเข้ารหัสการป้องกันที่พวกเขาบังคับใช้ ไม่น่าแปลกใจเลยที่เราพบว่าเทคนิคเหล่านี้บังคับใช้แนวทางปฏิบัติเหล่านี้หลายประการ ตารางที่ 4 สรุป สำหรับแต่ละเทคนิค ซึ่งแนวทางปฏิบัติในการเข้ารหัสเชิงป้องกันที่บังคับใช้

7.CONCLUSION

ในบทความนี้ เราได้นำเสนอการสำรวจ และ การเปรียบเทียบเทคนิคปัจจุบันในการตรวจจับ และ ป้องกัน SQLIAs เพื่อทำการประเมินนี้ อันดับแรก เราได้ระบุประเภทต่าง ๆ ของ SQLIAs ที่รู้จักจนถึงปัจจุบัน จากนั้นเราประเมินเทคนิคที่พิจารณาในแง่ของความสามารถในการตรวจจับ และ/หรือ ป้องกันการโจมตีดังกล่าว เรายังศึกษากลไกต่างๆ ที่ SQLIAs สามารถนำมาใช้ในแอปพลิเคชัน และ ระบุว่าเทคนิคใดบ้างที่สามารถจัดการกับกลไกต่างๆ ได้ สุดท้ายนี้ เราได้สรุปข้อกำหนดในการปรับใช้ของแต่ละเทคนิค และ ประเมินว่ากลไกการตรวจจับ และ การป้องกันของเทคนิคนั้นสามารถทำงานอัตโนมัติได้อย่างเต็มที่ในระดับใด

การประเมินของเราพบแนวโน้มทั่วไปหลายประการในผลลัพธ์ เทคนิคหลายอย่างมีปัญหาในการจัดการการโจมตีที่ใช้ประโยชน์จากขั้นตอนการจัดเก็บที่มีรหัสไม่ดี และ ไม่สามารถจัดการกับการโจมตีที่ปลอมตัวโดยใช้การเข้ารหัสสำรอง นอกจากนี้เรายังพบความแตกต่างทั่วไปในความสามารถในการป้องกันโดยพิจารณาจากความแตกต่างระหว่างเทคนิคการตรวจหา และ ป้องกันที่เน้นการป้องกัน และ ทั่วไป ส่วนที่ 6.4 เสนอว่าความแตกต่างนี้สามารถอธิบายได้ด้วยข้อเท็จจริงที่ว่าเทคนิคที่เน้นการป้องกันพยายามรวมแนวปฏิบัติที่ดีที่สุดในการเข้ารหัสการป้องกันไว้ในกลไกการป้องกันการโจมตี

งานประเมินในอนาคตควรเน้นที่การประเมินความแม่นยำ และ ประสิทธิผลของเทคนิคในทางปฏิบัติ การประเมินเชิงประจักษ์ เช่น ที่นำเสนอในงานที่เกี่ยวข้อง จะช่วยให้สามารถเปรียบเทียบประสิทธิภาพของเทคนิคต่างๆ เมื่อถูกโจมตีในโลกแห่งความเป็นจริงและข้อมูลป้อนเข้าที่ถูกต้อง

8.REFERENCES

[1] ค. แอนลีย์. SQL injectionv ขั้นสูงในแอปพลิเคชันเซิร์ฟเวอร์ SQL เอกสารไวท์เปเปอร์ Next Generation Security Software Ltd., 2002

[2] ซี. แอนลีย์. (เพิ่มเติม) SQL injection ขั้นสูง เอกสารไวท์เปเปอร์ Next Generation Security Software Ltd., 2002

[3] ดี. ออสสมิธ. การสร้าง และ บำรุงรักษาซอฟต์แวร์ที่ต่อต้านการโจมตีที่เป็นอันตราย http://www.gtisc.gatech.edu/bio aucsmith.html กันยายน 2547 Distinguished Lecture Series

[4] เอฟ. บูมา. กระบวนงานที่เก็บไว้ไม่ดี O'kay? รายงานทางเทคนิค Asp.Net Weblogs พฤศจิกายน 2546 http://weblogs.asp net/fbouma/archive/2003/11/18/38178.aspx.

[5] S.W. Boyd และ A.D. Keromytis. SQLrand: การป้องกันการโจมตีด้วยการฉีด SQL ในการดำเนินการของการประชุม Applied Cryptography and Network Security (ACNS) ครั้งที่ 2 หน้าที่ 292–302 มิถุนายน 2547
[6] G. T. Buehrer, B. W. Weide และ P. A. G. Sivilotti การใช้การตรวจสอบความถูกต้องของ Parse Tree เพื่อป้องกันการโจมตีของ SQL Injection ในการประชุมเชิงปฏิบัติการระหว่างประเทศด้านวิศวกรรมซอฟต์แวร์และมิดเดิลแวร์ (SEM), 2005
[7] ดับบลิว อาร์ คุก และ ส. ไร่. Safe Query Objects: Objects ที่พิมพ์แบบสแตติกเป็นคิวรีที่ดำเนินการได้จากระยะไกล ในการดำเนินการของการประชุมนานาชาติด้านวิศวกรรมซอฟต์แวร์ครั้งที่ 27 (ICSE 2005), 2005
[8] เอ็ม. ดอร์นเซอิฟ. ความล้มเหลวทั่วไปในแอปพลิเคชันอินเทอร์เน็ต พฤษภาคม 2005 http://md.hudora.de/presentations/2005-common-failures/ dornseif-common-failures-2005-05-25.pdf
[9] อี.เอ็ม.ฟาโย. การฉีด SQL ขั้นสูงในฐานข้อมูล Oracle รายงานทางเทคนิค Argeniss Information Security, Black Hat Briefings, Black Hat USA, 2005
[10] พี. ฟินนิแกน. SQL Injection และ Oracle - ส่วนที่ 1 และ 2 รายงานทางเทคนิค, Security Focus, พฤศจิกายน 2002
http://securityfocus.com/infocus/1644
http://securityfocus.com/infocus/1646
[11] มูลนิธิ ที.โอ. ช่องโหว่ของเว็บแอปพลิเคชันที่สำคัญที่สุด 10 อันดับแรก พ.ศ. 2548 http: //www.owasp.org/documentation/topten.html
[12] C. Gould, Z. Su และ P. Devanbu JDBC Checker: เครื่องมือวิเคราะห์แบบคงที่สำหรับแอปพลิเคชัน SQL/JDBC ในการดำเนินการของการประชุมนานาชาติด้านวิศวกรรมซอฟต์แวร์ครั้งที่ 26 (ICSE 04) – การสาธิตอย่างเป็นทางการ หน้า 697–698, 2004
[13] C. Gould, Z. Su และ P. Devanbu การตรวจสอบแบบคงที่ของแบบสอบถามที่สร้างขึ้นแบบไดนามิกในแอปพลิเคชันฐานข้อมูล ในการดำเนินการประชุมนานาชาติด้านวิศวกรรมซอฟต์แวร์ครั้งที่ 26 (ICSE 04), หน้า 645–654, 2004
[14] N. W. กลุ่ม. RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1 ขอความคิดเห็น The Internet Society, 1999.

[15] V. Haldar, D. Chandra และ M. Franz การขยายพันธุ์แบบไดนามิกสำหรับ Java ในการประชุมประจำปีของการประชุม Computer Security Applications ครั้งที่ 21 ธันวาคม พ.ศ. 2548
[16] W. G. Halfond และ A. Orso. ความจำเสื่อม: การวิเคราะห์และการตรวจสอบสำหรับการทำให้การโจมตีด้วยการฉีด SQL เป็นกลาง ในการดำเนินการของ IEEE และ ACM International Conference on Automated Software Engineering (ASE 2005), Long Beach, CA, USA, พ.ย. 2548 ที่จะปรากฏ
[17] W. G. Halfond และ A. Orso. การรวมการวิเคราะห์แบบสถิตและการตรวจสอบรันไทม์เพื่อตอบโต้การโจมตีด้วยการฉีด SQL In Proceedings of the Third International ICSE Workshop on Dynamic Analysis (WODA 2005), หน้าที่ 22–28, St. Louis, MO, USA, พฤษภาคม 2005
[18] M. Howard และ D. LeBlanc. การเขียนรหัสที่ปลอดภัย Microsoft Press, Redmond, Washington, ฉบับที่สอง, 2003
.[19] Y. Huang, S. Huang, T. Lin และ C. Tsai. การประเมินความปลอดภัยของเว็บแอปพลิเคชันโดยการฉีดข้อผิดพลาดและการตรวจสอบพฤติกรรม ในการดำเนินการของการประชุมนานาชาติเวิลด์ไวด์เว็บครั้งที่ 11 (WWW 03) พฤษภาคม 2546
[20] Y. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee และ S. Y. Kuo การรักษาความปลอดภัยรหัสแอปพลิเคชันเว็บโดยการวิเคราะห์แบบคงที่และการป้องกันรันไทม์ ในการดำเนินการของการประชุมนานาชาติเวิลด์ไวด์เว็บครั้งที่ 12 (WWW 04) พฤษภาคม 2547
[21] เอส. แล็บ. การฉีด SQL เอกสารไวท์เปเปอร์, SPI Dynamics, Inc., 2002 http://www.spidynamics.com/assets/documents/WhitepaperSQLInjection.pdf
[22] ง. ทุ่งลิ้นจี่. การถอดประกอบเว็บแอปพลิเคชันพร้อมข้อความแสดงข้อผิดพลาด ODBC เอกสารทางเทคนิค @Stake, Inc., 2002 http://www.nextgenss.com/papers/webappdis.doc
[23] V. B. Livshits และ M. S. Lam การค้นหาข้อผิดพลาดด้านความปลอดภัยในโปรแกรม Java ด้วยการวิเคราะห์แบบคงที่ ในการดำเนินการของ Usenix Security Symposium ครั้งที่ 14 หน้า 271–286 ส.ค. 2548
[24] ซี.เอ.แมคเคย์. การโจมตีด้วยการฉีด SQL และเคล็ดลับบางประการเกี่ยวกับวิธีการป้องกัน รายงานทางเทคนิค The Code Project มกราคม 2548 http://www.codeproject.com/cs/database/ SqlInjectionAttacks.asp

[25] O. Maor and A. Shulman. SQL Injection Signatures Evasion. White paper, Imperva, April 2004. http://www.imperva.com/ application defense center/white papers/ sql injection signatures evasion.html.
[26] M. Martin, B. Livshits และ M. S. Lam การค้นหาข้อผิดพลาดของแอปพลิเคชันและข้อบกพร่องด้านความปลอดภัยโดยใช้ PQL: A Program Query Language ในการดำเนินการของการประชุม ACM SIGPLAN ประจำปีครั้งที่ 20 เกี่ยวกับภาษาและแอปพลิเคชันระบบการเขียนโปรแกรมเชิงวัตถุ (OOPSLA 2005) หน้า 365–383, 2005
[27] R. McClure และ I. Kr¨uger. SQL DOM: การตรวจสอบเวลาคอมไพล์ของคำสั่ง Dynamic SQL ในการดำเนินการของการประชุมนานาชาติด้านวิศวกรรมซอฟต์แวร์ครั้งที่ 27 (ICSE 05), หน้า 88–96, 2005
[28] เอส. แมคโดนัลด์. การฉีด SQL: โหมดการโจมตี การป้องกัน และเหตุใดจึงสำคัญ เอกสารไวท์เปเปอร์ GovernmentSecurity.org เมษายน 2545http://www.governmentsecurity.org/articles/ SQLInjectionModesofAttackDefenceandWhyIt Matters.php
[29] เอส. แมคโดนัลด์. เกมส์ฉีด SQL. เอกสารไวท์เปเปอร์ SecuriTeam พฤษภาคม 2002 http://www.securiteam.com/ securityreviews/5DP0N1P76E.html
[30] ที.เอ็ม.ดี.เครือข่าย Request.servervariables คอลเลกชัน รายงานทางเทคนิค Microsoft Corporation, 2005 http://msdn.microsoft.com/library/default asp?url=/library/en-us/iissdk/html/ 9768ecfe-8280-4407-b9c0-844f75508752.asp.
[31] A. Nguyen-Tuong, S. Guarnieri, D. Greene, J. Shirley และ D. Evans การทำให้เว็บแอปพลิเคชันแข็งตัวโดยอัตโนมัติโดยใช้ข้อมูลการย้อมสีที่แม่นยำ ในการประชุม IFIP International Information Security Conference ครั้งที่ 20 (SEC 2005) พฤษภาคม 2548
[32] T. Pietraszek และ C.V. Berghe. การป้องกันการโจมตีด้วยการฉีดผ่านการประเมินสตริงที่มีความละเอียดอ่อนตามบริบท ในการดำเนินการเกี่ยวกับความก้าวหน้าล่าสุดในการตรวจจับการบุกรุก (RAID2005), 2005
[33] ดี. สก็อตต์ และ อาร์. ชาร์ป บทคัดย่อการรักษาความปลอดภัยเว็บระดับแอปพลิเคชัน ในการดำเนินการประชุมนานาชาติครั้งที่ 11 บนเวิลด์ไวด์เว็บ (WWW 2002) หน้า 396–407, 2002

[34] K. Spett. Blind sql injection. White paper, SPI Dynamics, Inc., 2003. http://www.spidynamics.com/whitepapers/ Blind SQLInjection.pdf.
[35] Z. Su และ G. Wassermann. แก่นแท้ของการโจมตีด้วย Command Injection ในเว็บแอปพลิเคชัน ในการประชุมวิชาการประจำปีครั้งที่ 33 เรื่อง หลักการของภาษาการเขียนโปรแกรม (POPL 2006), ม.ค. 2549
[36] F. Valeur, D. Mutz และ G. Vigna แนวทางการเรียนรู้ที่เป็นฐานในการตรวจหาการโจมตีของ SQL ในการดำเนินการประชุมว่าด้วยการตรวจจับการบุกรุกและมัลแวร์และการประเมินช่องโหว่ (DIMVA), เวียนนา, ออสเตรีย, กรกฎาคม 2548
[37] G. Wassermann และ Z. Su. กรอบงานการวิเคราะห์เพื่อความปลอดภัยในเว็บแอปพลิเคชัน ในการดำเนินการของ FSE Workshop on Specificiation and Verificiation of Component-Based Systems (SAVCBS 2004), หน้า 70–78, 2004