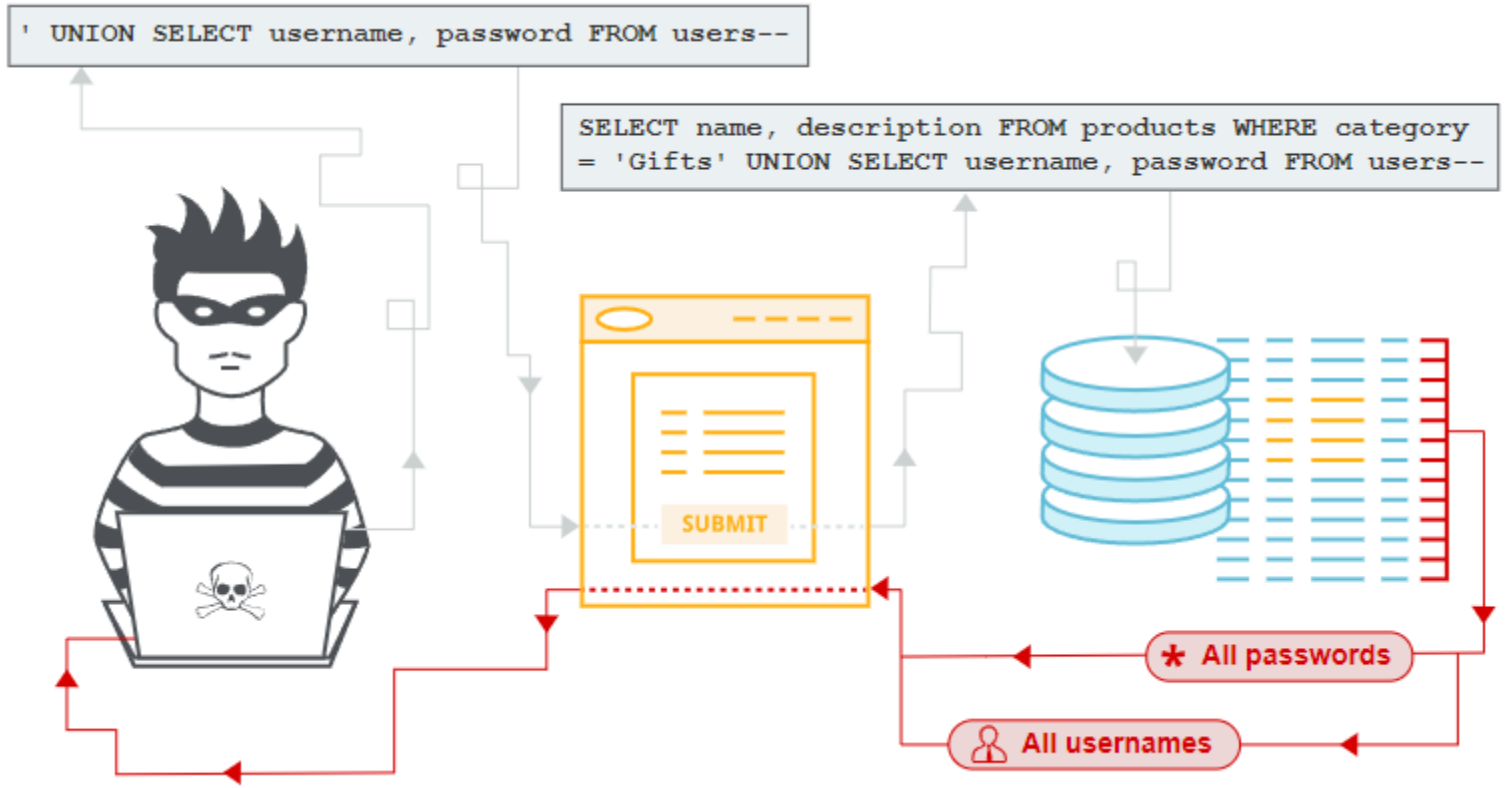

e-Cop®
Managed Security Service Provider

e-Cop®

Example of an incident

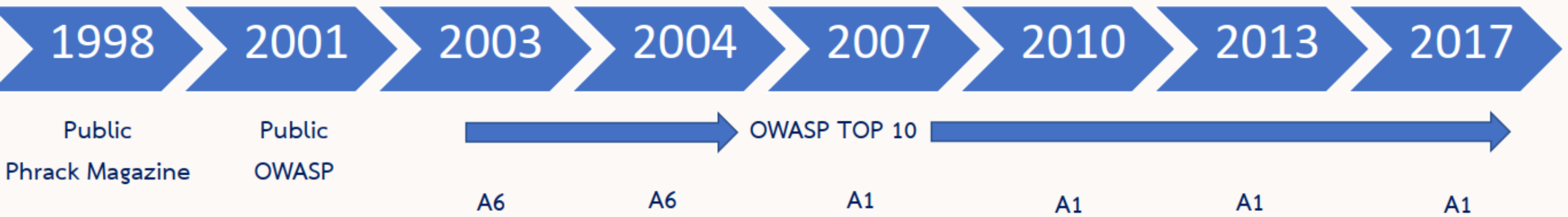
Example of an incident



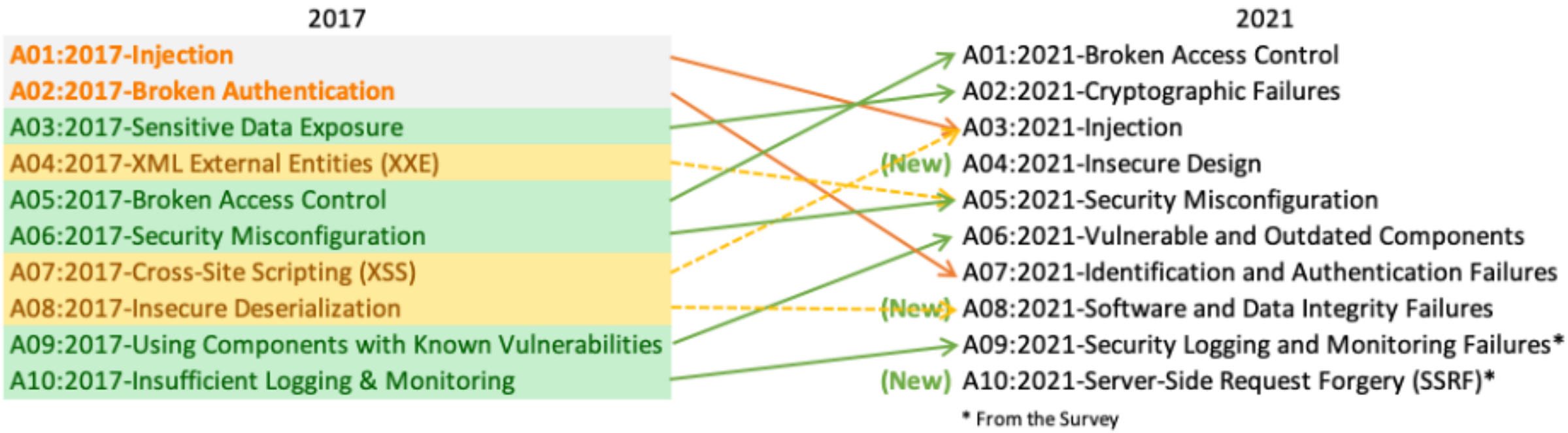
หัวข้อในการบรรยาย

- SQL Injection Timeline
- What is SQL injection
- Types of SQL Injection (SQLi)
- แนวทางการป้องกัน SQL Injection
- NIST Cybersecurity Framework

SQL Injection Timeline



SQL Injection เกิดขึ้นมา 21 ปี


2017
A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring
2021
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*
* From the Survey

A3 2021

What is SQL injection

- SQL injection (SQLi) คือ รูปแบบการโจมตีเทคนิคหนึ่งในหัวข้อ Injection ที่ Attacker สามารถอาศัยช่องโหว่ของการประมวลผลที่ไม่มีการตรวจสอบความถูกต้องของข้อมูลนำเข้า (input validation) เช่น ทำให้ Attacker นั้นสามารถแทรกคำสั่ง SQL เพื่อเปลี่ยนแปลงการทำงานของการประมวลผลบนฐานข้อมูลให้เป็นไปตามความต้องการของ Hacker ได้สำเร็จ เช่น เพิ่มข้อมูล (Insert), อัพเดตข้อมูล (Update) ลบข้อมูล (Delete) เป็นต้น

คำสั่งภาษา SQL

- SQL มาจากคำว่า Structured Query Language เป็นภาษามาตรฐานกลางในการบริหารจัดการฐานข้อมูล เช่น SQL Server, MySQL, Oracle เป็นต้น โดยมีรูปแบบของคำสั่งมาตรฐานที่ถูกกำหนดโดย ANSI (American National Standard Institute) ซึ่งมีรูปแบบของคำสั่งที่ง่ายต่อการทำงาน

SQL commands

SELECT

WHERE

FROM

DELETE

INSERT

UPDATE

Special Characters

single quote (‘)

double quote (“)

equal (=)

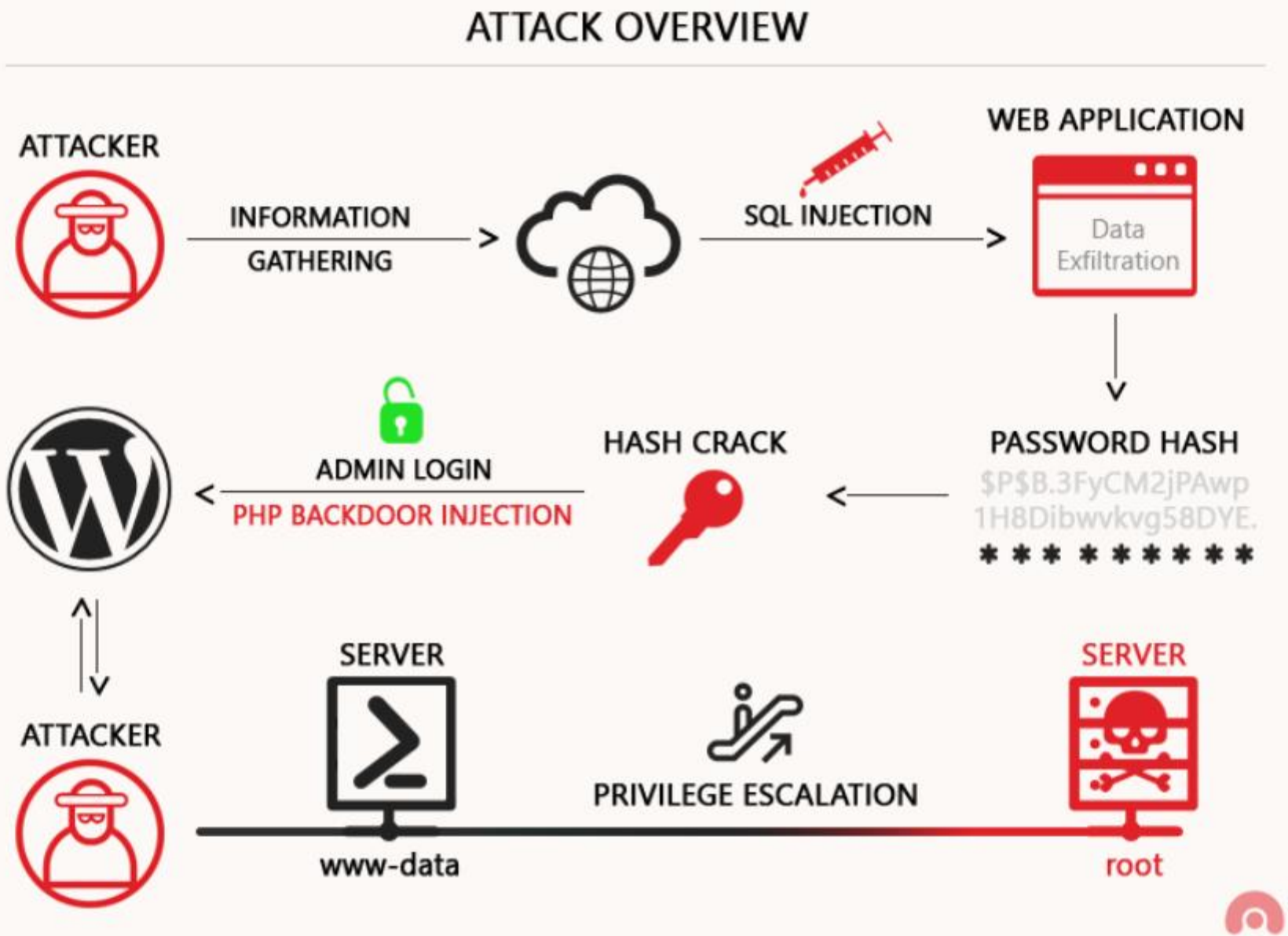
comment (- -,#,/*,*/)

SQL AND, OR Operators

ประเภทของ SQL query

- Select query ใช้ดึงข้อมูลในรูป table คือเป็น row และ column
  - SELECT column1, column2, ...
  - FROM table_name;
- Update query ใช้แก้ไขข้อมูลที่มีอยู่แล้วใน table
  - UPDATE table_name
  - SET column1 = value1, column2 = value2, ...
  - WHERE condition;
- Insert query ใช้เพิ่มข้อมูลใน table
  - INSERT INTO table_name (column1, column2, column3, ...)
  - VALUES (value1, value2, value3, ...);
- Delete query ใช้ลบข้อมูลใน table
  - DELETE FROM table_name
  - WHERE condition;

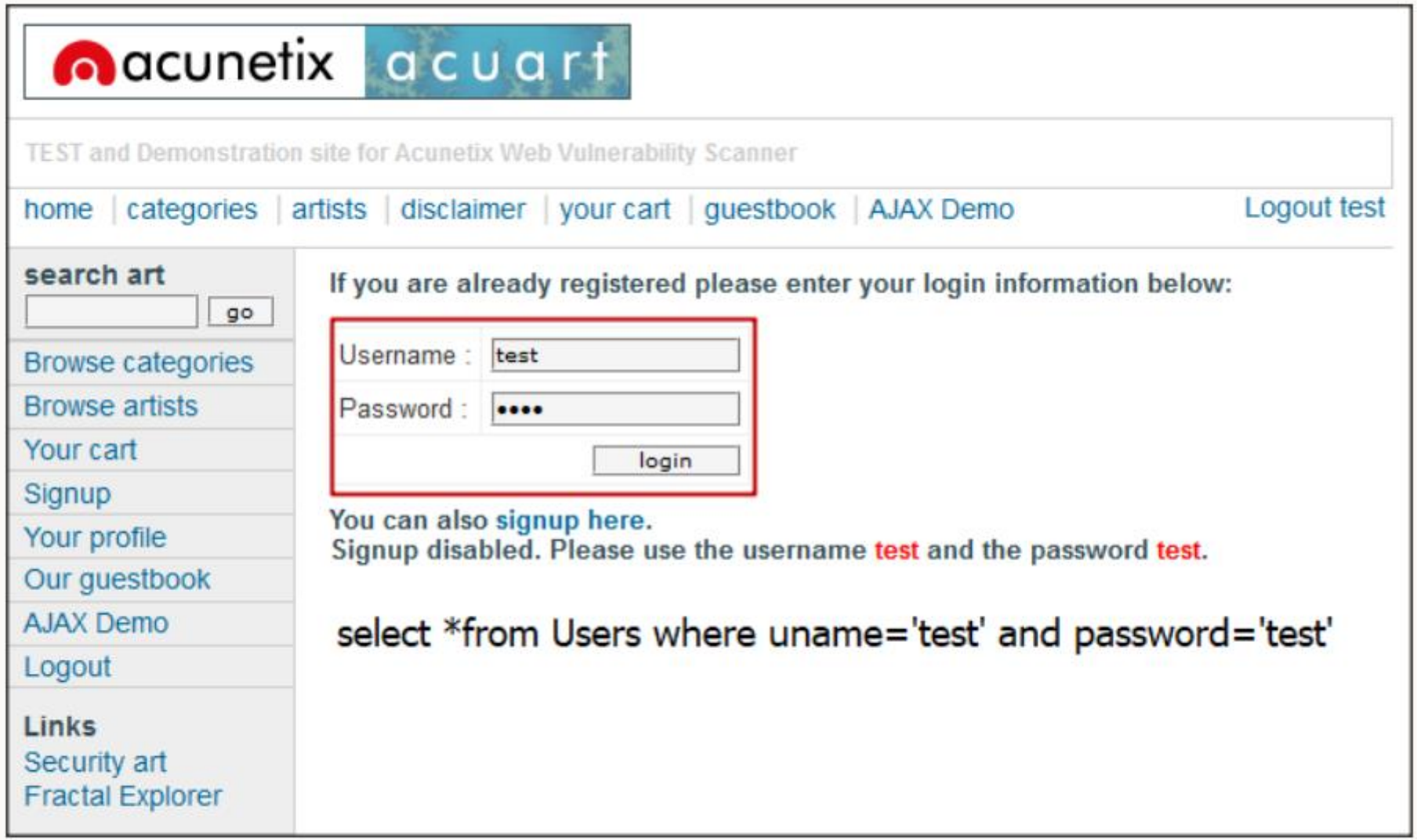
SQL injection: Scenario



SQL injection: Login Bypass Example

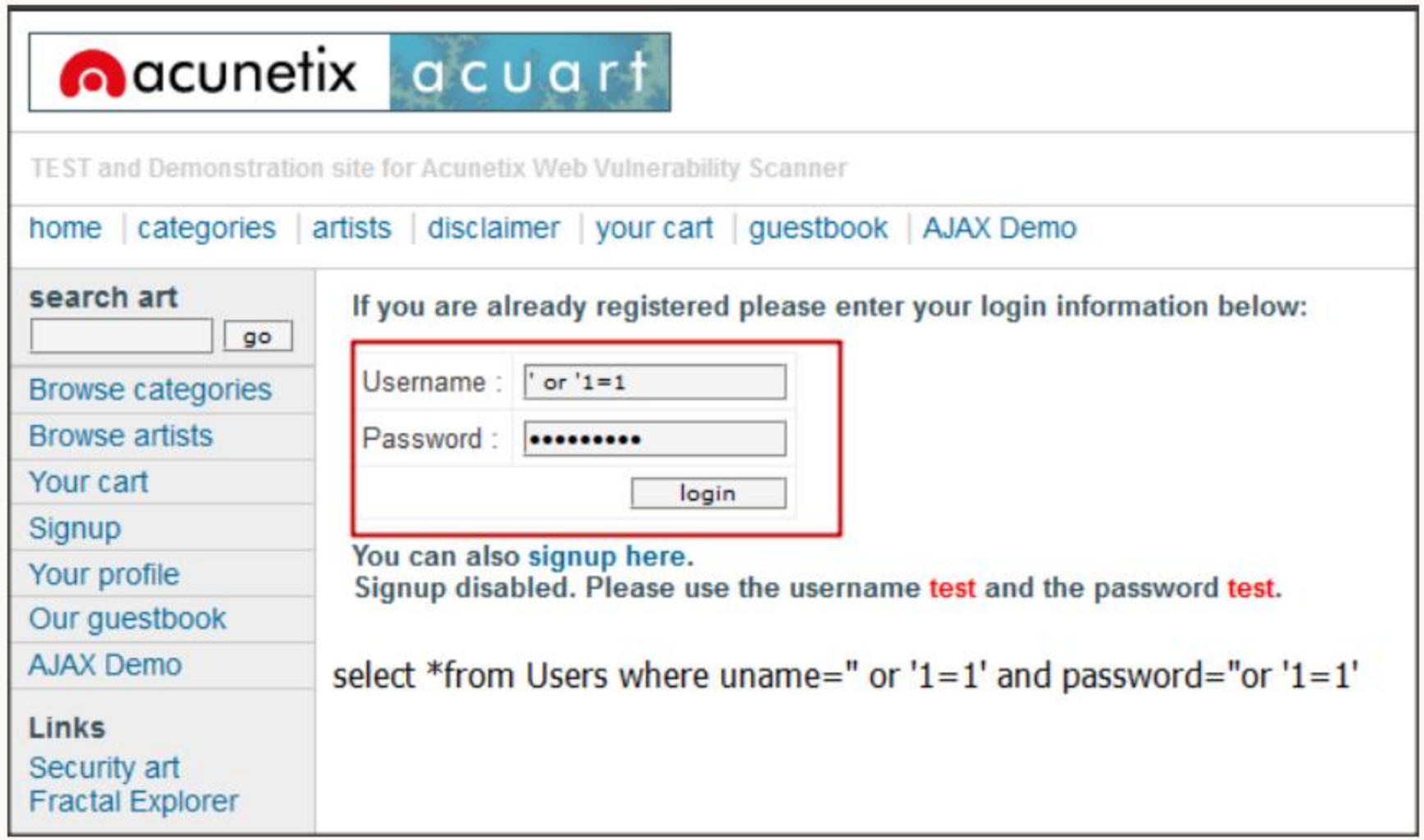
- ค้นหาหน้า Login
  - inurl:/login.php
  - inurl:/admin.php
  - inurl:/admin
  - inurl:/login.html
- ทดสอบ
  - Username: admin
  - Password: ' or 0=0 #

SQL injection: Login Bypass Example



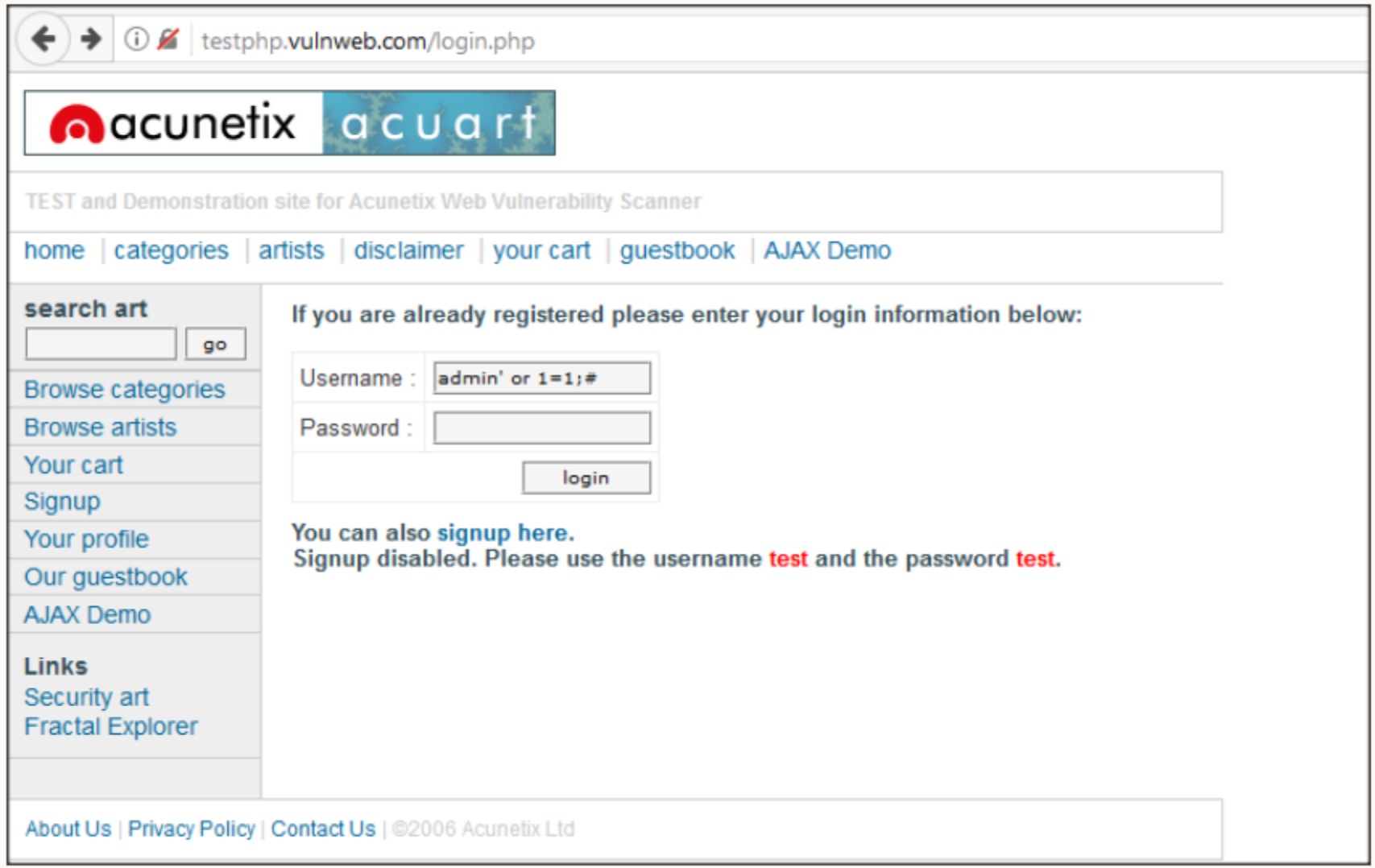
select *from Users where uname='test' and password='test'

SQL injection: Login Bypass Example



select *from Users where uname=" or '1=1' and password="or '1=1'

SQL injection: Login Bypass Example



select *from Users where uname='admin' or 1=1;#' and password=''

SQL injection: Login Bypass Example

User name	Password	SQL Query
tom	tom	SELECT * FROM users WHERE name='tom' and password='tom'
tom	' or '1'='1	SELECT * FROM users WHERE name='tom' and password='' or '1'='1'
tom	' or 1='1	SELECT * FROM users WHERE name='tom' and password='' or 1='1'
tom	1' or 1=1 -- -	SELECT * FROM users WHERE name='tom' and password='' or 1=1-- -'
' or '1'='1	' or '1'='1	SELECT * FROM users WHERE name='' or '1'='1' and password='' or '1'='1'
' or ' 1=1	' or ' 1=1	SELECT * FROM users WHERE name='' or ' 1=1' and password='' or ' 1=1'
1' or 1=1 --	blah	SELECT * FROM users WHERE name='1' or 1=1 -- -' and password='blah'

Types of SQL Injection (SQLi)

In-band SQLi (Classic SQLi)

- In-band SQL Injection is the most common and easy-to-exploit of SQL Injection attacks. In-band SQL Injection occurs when an attacker is able to use the same communication channel to both launch the attack and gather results.

- The two most common types of in-band SQL Injection are Error-based SQLi and Union-based SQLi.

Types of SQL Injection (SQLi)

Error-based SQLi

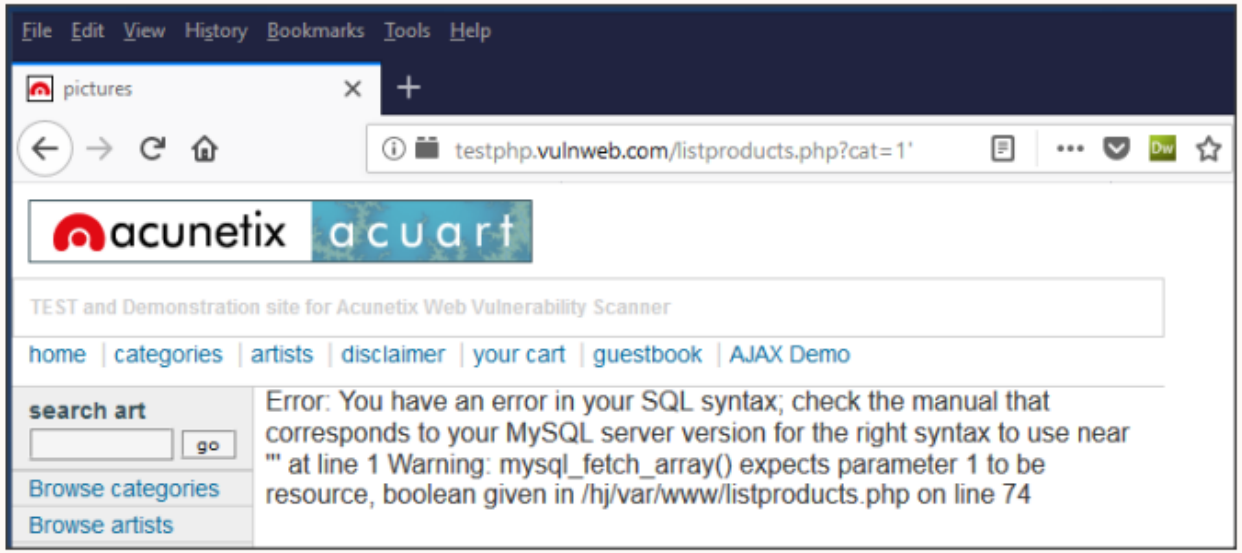
- Error-based SQLi is an in-band SQL Injection technique that relies on error messages thrown by the database server to obtain information about the structure of the database. In some cases, error-based SQL injection alone is enough for an attacker to enumerate an entire database. While errors are very useful during the development phase of a web application, they should be disabled on a live site, or logged to a file with restricted access instead

SQL injection: Error-based Example

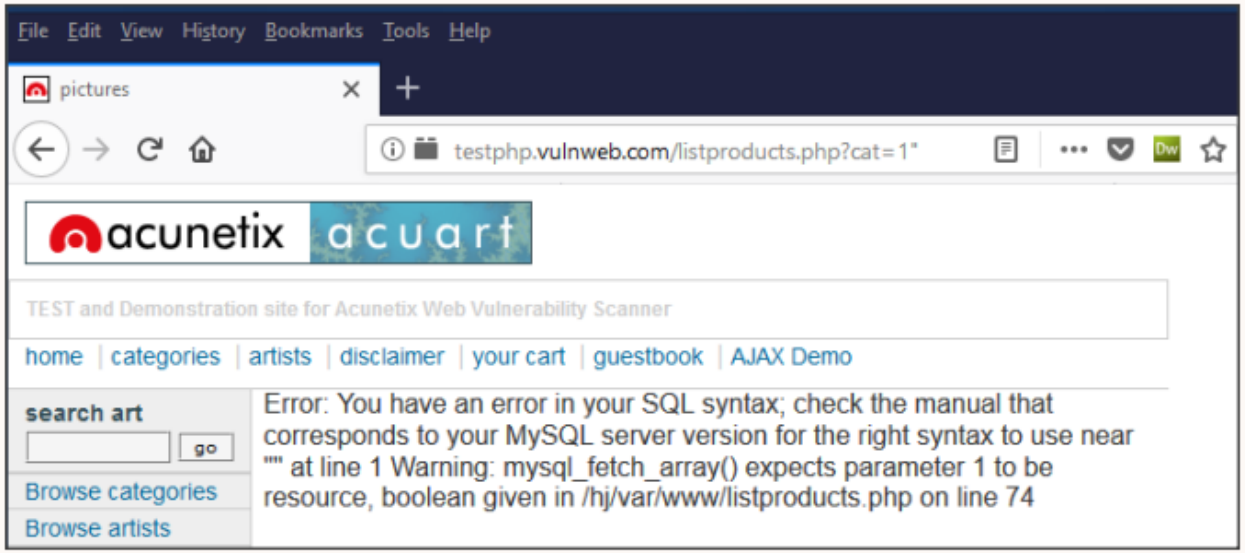
- คำค้นหา SQL
  - inurl:index.php?id=
  - inurl:gallery.php?id=
  - inurl:article.php?id=
  - inurl:pageid.php?id=
- ทดสอบ
  - site:tot.co.th inurl:php?id=
  - site:tot.co.th inurl:index.php?id=

SQL injection: Error-based Example

single quote (')

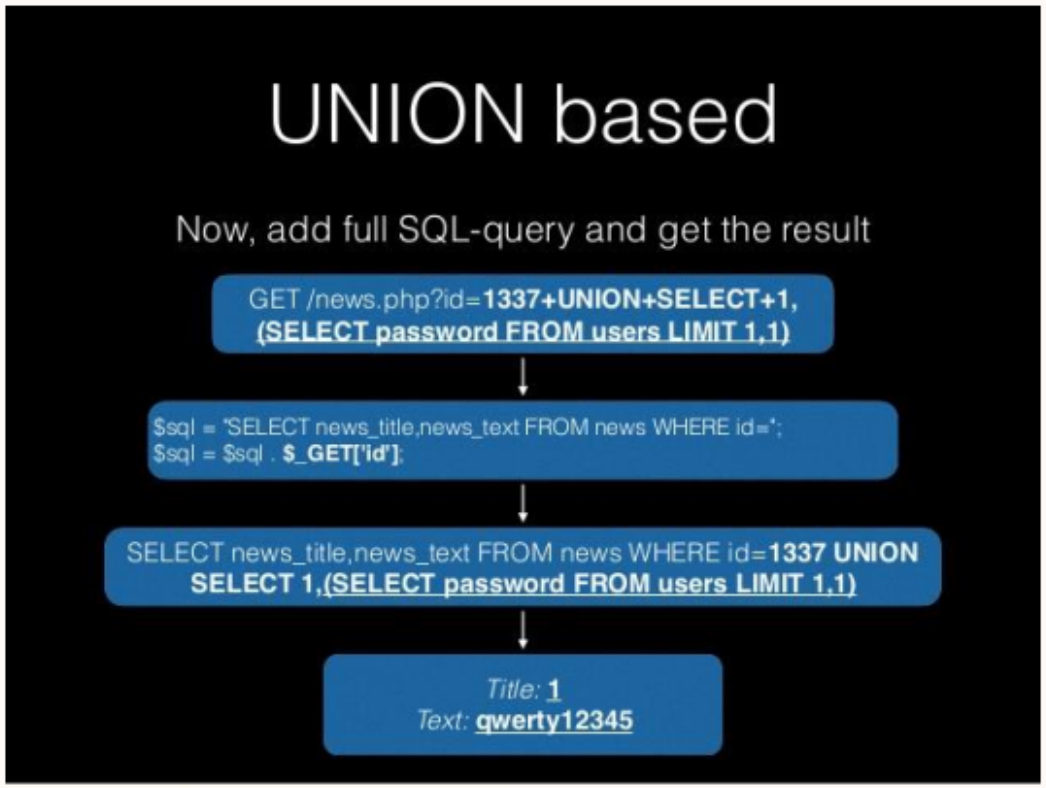


double quote (")



Union-based SQLi

- Union-based SQLi is an in-band SQL injection technique that leverages the UNION SQL operator to combine the results of two or more SELECT statements into a single result which is then returned as part of the HTTP response.



SQL injection: Union-based Example

- http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1, 2, 3

Types of SQL Injection (SQLi)

- Inferential SQLi (Blind SQLi)
- Inferential SQL Injection, unlike in-band SQLi, may take longer for an attacker to exploit, however, it is just as dangerous as any other form of SQL Injection. In an inferential SQLi attack, no data is actually transferred via the web application and the attacker would not be able to see the result of an attack in-band (which is why such attacks are commonly referred to as “blind SQL Injection attacks”). Instead, an attacker is able to reconstruct the database structure by sending payloads, observing the web application’s response and the resulting behavior of the database server.
- The two types of inferential SQL Injection are Blind-boolean-based SQLi and Blind-time-based SQLi.

Boolean-based (content-based) Blind SQLi

- Boolean-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the application to return a different result depending on whether the query returns a TRUE or FALSE result.

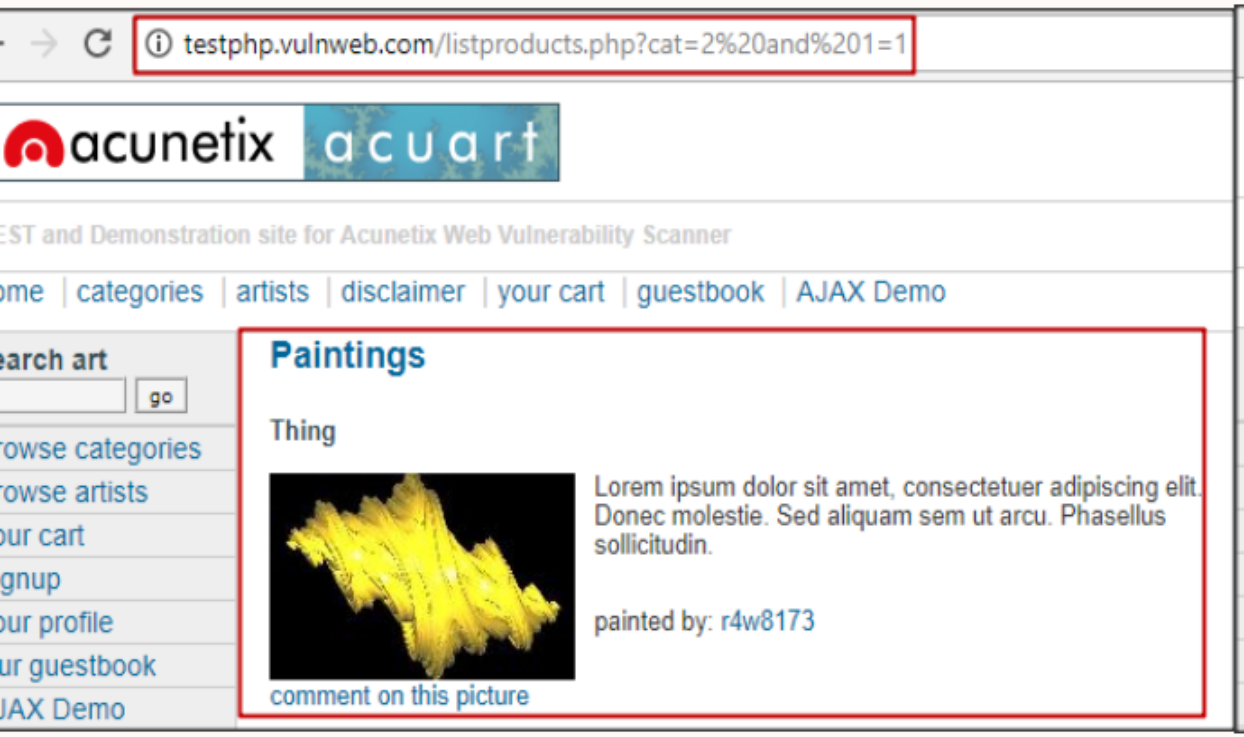
- Depending on the result, the content within the HTTP response will change, or remain the same. This allows an attacker to infer if the payload used returned true or false, even though no data from the database is returned. This attack is typically slow (especially on large databases) since an attacker would need to enumerate a database, character by character.

SQL injection: Boolean-based Example

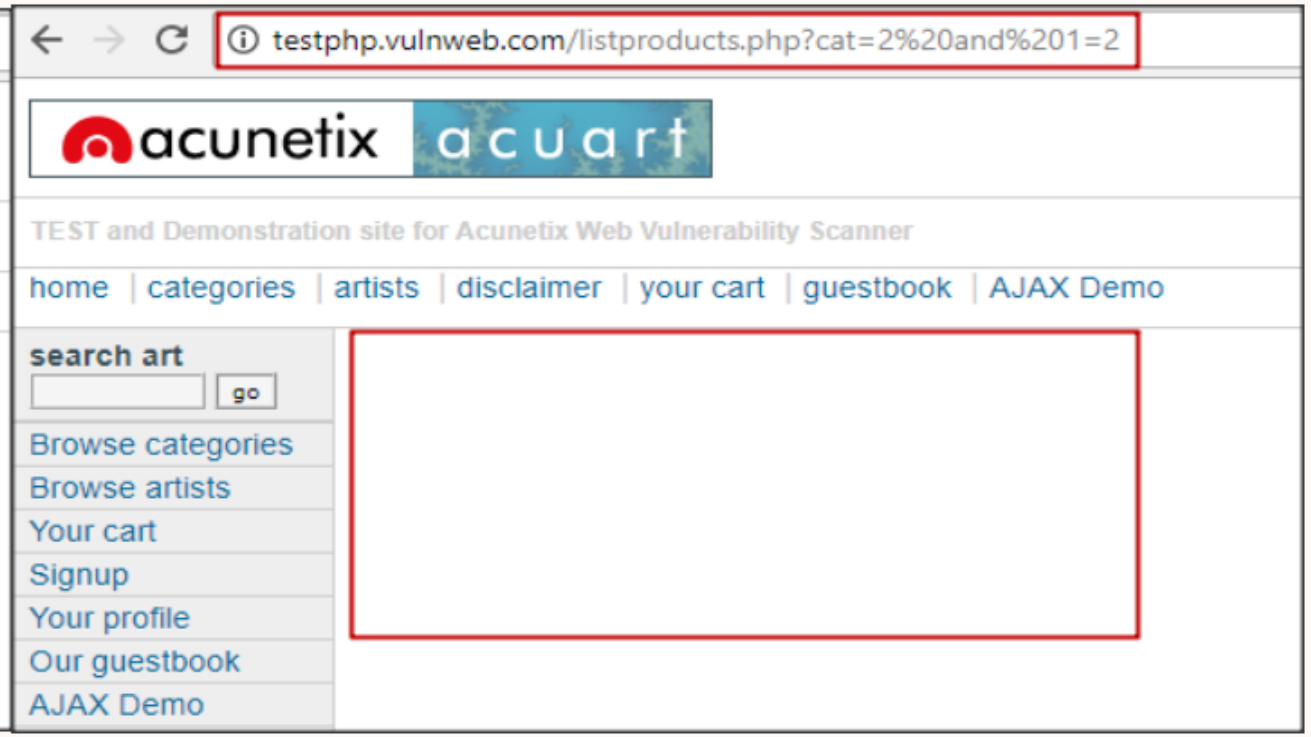
- http://www.example.com/item.php?id=34
- SELECT name, description, price FROM Store_table WHERE id = 34
- http://www.example.com/item.php?id=34 and 1=2
- SELECT name, description, price FROM Store_table WHERE id = 34 and 1=2
- Return FALSE
- http://www.example.com/item.php?id=34 and 1=1
- SELECT name, description, price FROM Store_table WHERE ID = 34 and 1=1
- Return TRUE

SQL injection: Boolean-based Example

http://testphp.vulnweb.com/listproducts.php?cat=2 and 1=1



http://testphp.vulnweb.com/listproducts.php?cat=2 and 1=2



Time-based Blind SQLi

- Time-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the database to wait for a specified amount of time (in seconds) before responding. The response time will indicate to the attacker whether the result of the query is TRUE or FALSE.

- Depending on the result, an HTTP response will be returned with a delay, or returned immediately. This allows an attacker to infer if the payload used returned true or false, even though no data from the database is returned. This attack is typically slow (especially on large databases) since an attacker would need to enumerate a database character by character.

Time-based Blind SQLi Example

MySQL Time-Based Attack

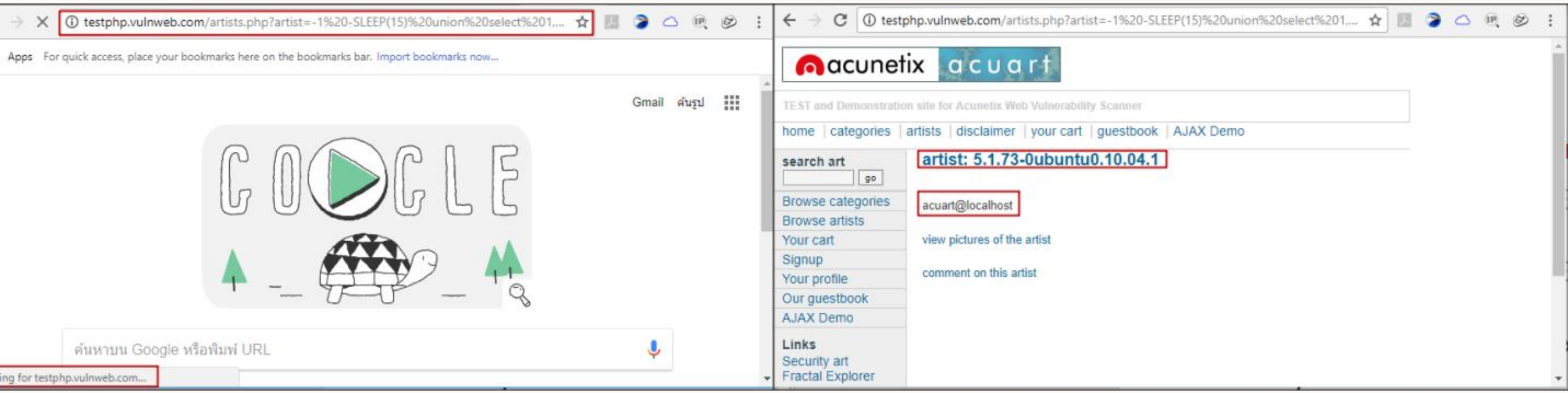
- SELECT * FROM products WHERE id=1-SLEEP(15)
- SELECT * FROM products WHERE id=1-BENCHMARK(100000000, rand())
- SELECT * FROM products WHERE id=1-IF(MID(VERSION(),1,1) = '5', SLEEP(15), 0)

SQL Server Time-Based

- SELECT * FROM products WHERE id=1; WAIT FOR DELAY '00:00:15’
- SELECT * FROM products WHERE id=1; IF SYSTEM_USER='sa' WAIT FOR DELAY '00:00:15'

Time-based Blind SQLi Example

- http://testphp.vulnweb.com/artists.php?artist=-1 -SLEEP(15) union select 1,version(),current_user()



Workshop

แนวทางการป้องกัน SQL Injection

แนวทางการป้องกัน SQL Injection

แนวทางการป้องกัน SQL Injection ในการพัฒนาโปรแกรมด้านเว็บแอปพลิเคชันสามารถแบ่งออกเป็น 2 วิธี ดังนี้

- Use of Prepared Statements (with Parameterized Queries)
- Use of Stored Procedures
- เพิ่มเติม
- Enforcing Least Privilege
- Performing Whitelist Input Validation as a Secondary Defense

Use of Prepared Statements

- Prepared statement or parameterized statement is a feature used to execute the same or similar database statements repeatedly with high efficiency. Typically used with SQL statements such as queries or updates, the prepared statement takes the form of a template into which certain constant values are substituted during each execution

Support Prepared statement

MySQL,Oracle,DB2,Microsoft SQL Server and PostgreSQL

Use of Prepared Statements

- Prepare: At first, the application creates the statement template and send it to the DBMS. Certain values are left unspecified, called parameters, placeholders or bind variables (labelled "?" below):
  - INSERT INTO products (name, price) VALUES (?, ?);
- Then, the DBMS compiles (parses, optimizes and translates) the statement template, and stores the result without executing it.
- Execute: At a later time, the application supplies (or binds) values for the parameters of the statement template, and the DBMS executes the statement (possibly returning a result). The application may execute the statement as many times as it wants with different values. In the above example, it might supply "bike" for the first parameter and "10900" for the second parameter.

Use of Prepared Statements

- PHP PDO คืออะไรวันนี้มีคำตอบนะครับ PDO คือ Extension หรือส่วนเสริมของ PHP ย่อมาจากคำว่า PHP DataObject ซึ่งเป็น Object ที่ใช้ในการเชื่อมต่อกับฐานข้อมูลได้หลากหลาย

Warning This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the MySQLi or PDO_MySQL extension should be used. See also MySQL: choosing an API guide and related FAQ for more information. Alternatives to this function include:

- mysqli_connect()
- PDO::__construct()

PHP เวอร์ชั่น 5.5 ขึ้นไปได้ยกเลิกการเชื่อมต่อ MySQL แบบเดิมแล้วคือการใช้ Function mysql_connect() และ mysql_xxxx() ทั้งหลาย แต่จะไปใช้ MySQLi หรือ MySQL Improved แทนซึ่งมีความปลอดภัยมากกว่า

Use of Prepared Statements

- //mysql_connect
- $link = mysql_connect('localhost', 'user', 'password');
- mysql_select_db('dbname', $link);
- //mysqli
- $link = mysqli_connect('localhost', 'user', 'password', 'dbname');
- //PDO
- $PDO = new PDO("mysql:host=localhost;dbname=database;charset=UTF8", userhost, passhost);

Safe PHP Prepared Statement Example

- # การติดต่อฐานข้อมูล

```
$dbhost = 'localhost';
$dbuser='root';
$dbpass='‘123456';
$pdo = new PDO("mysql:dbname={$dbname};host=($dbhost}", $dbuser, $dbpass);
```

- # การคิวรี่ข้อมูล

```
$result = $pdo->prepare("SELECT * FROM table_name");
$result->execute();
while($rs = $result->fetch()){
    echo $rs['field_name'];
}
```

Safe PHP Prepared Statement Example

- # การเรียกข้อมูล 1 แถว

```
$sql = "SELECT * FROM table WHERE field_name = :field_param";
$result = $pdo->prepare($sql);
$result->execute(array(':field_param'=>$_GET['param']));
```

- การทำการ bind param นั้นเป็นการแทนที่ :field_param ด้วย ค่าที่ $_GET['param']
- # การเพิ่มข้อมูล

```
$sql = "INSERT INTO table_name('field1','field2') VALUES(:f1_param,:f2_param)";
$result = $pdo->prepare($sql);
$result->execute(array(
    ':f1_param'=>$_POST['param1'],
    'f2_param'=>$_POST['param2']
));
```

Safe PHP Prepared Statement Example

```
<?php
        $mysqli = new mysqli("localhost", "username", "password", "database_name");
        $limit = $mysqli->prepare("SELECT * FROM users WHERE username = ? AND password = ?");
        // ให้ ?  ใน query เป็นตัวแปรที่มาจากข้างนอก
        // กำหนดให้ ? เป็นชนิด string และ "เชื่อม" (bind) เข้ากับตัวแปร  $limit // s - string, b - blob, i – integer, d - double
        $limit->bind_param("ss", $username,$password);
        $username = $_POST['user_username'];
        $password = $_POST['user_password'];
        $limit->execute(); // ทำการรัน query
        if ($limit->fetch()){
                        echo "Login Success";
          }else{
             echo "Login Fail";
    }?>
```

SQL injection: JAVA Example

```
String query = "SELECT account_balance FROM user_data WHERE user_name = "
            + request.getParameter("customerName");
try {
            Statement statement = connection.createStatement( ... );
           ResultSet results = statement.executeQuery( query );
}
```

Use of Prepared Statements

- Language specific recommendations:
- Java EE – use PreparedStatement() with bind variables
- .NET – use parameterized queries like SqlCommand() or OleDbCommand() with bind variables
- PHP – use PDO with strongly typed parameterized queries (using bindParam())
- Hibernate - use createQuery() with bind variables (called named parameters in Hibernate)
- SQLite - use sqlite3_prepare() to create a statement object

Safe Java Prepared Statement Example

```
// This should REALLY be validated too
String custname = request.getParameter("customerName");
// Perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";
PreparedStatement pstmt = connection.prepareStatement(query);
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery();
```

Safe C# .NET Prepared Statement Example

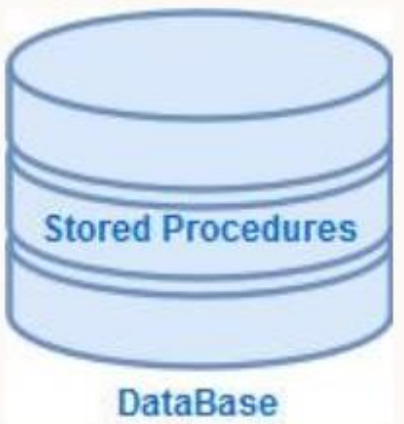
```
String query = "SELECT account_balance FROM user_data WHERE user_name = ?";
try {
        OleDbCommand command = new OleDbCommand(query, connection);
        command.Parameters.Add(new OleDbParameter("customerName",
         CustomerName Name.Text));
        OleDbDataReader reader = command.ExecuteReader();
        } catch (OleDbException se) {
// error handling
    }
```

Use of Prepared Statements (อ้างอิง)

- JAVA Prepared Statements
  - https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html
- .NET Prepared Statements
  - https://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.prepare(v=vs.110).aspx
- PHP Prepared Statements
  - http://php.net/manual/en/book.mysqli.php

Use of Stored Procedures

- Stored Procedures คือ การสร้างคำสั่ง SQL query ในรูปแบบ Database object ที่ถูกจัดเก็บไว้บนฐานข้อมูลต่างๆ เช่น MySQL, Microsoft SQL Server, Oracle เป็นต้น เพื่อทดแทนการรับคำสั่ง SQL query จากเว็บแอพพลิเคชันโดยตรงที่รวมไว้ในค่า String เดียวกันเพื่อนำไปประมวลผลบนฐานข้อมูลซึ่งทำให้เกิดช่องโหว่ SQL Injection ดังนั้นการทำ Stored Procedures ซึ่งจะรับเฉพาะค่า Parameters ที่จำเป็นจากเว็บแอพพลิเคชันไปยังฐานข้อมูลเพื่อนำไปประมวลผลบนฐานข้อมูลทำตามคำสั่งที่ถูกกำหนดไว้ใน Stored Procedure



MySQL Stored Procedure in PHP

- Stored Procedure Name : insertCustomer()

```
01. DROP PROCEDURE IF EXISTS insertCustomer;
02.
03. DELIMITER //
04. CREATE PROCEDURE insertCustomer(IN pCustomerID VARCHAR(4),
05.       IN pName VARCHAR(150),
06.       IN pEmail VARCHAR(150),
07.       IN pCountryCode VARCHAR(2),
08.       IN pBudget DECIMAL(18,2),
09.       IN pUsed DECIMAL(18,2))
10. BEGIN
11.     INSERT INTO customer (CUSTOMER_ID, NAME, EMAIL, COUNTRY_CODE, BUDGET, USED)
12.       VALUES (pCustomerID, pName, pEmail, pCountryCode, pBudget, pUsed);
13. END //
14. DELIMITER ;
```

MySQL Stored Procedure in PHP

- add.php

```
01. <html>
02. <head>
03. <title>ThaiCreate.Com PHP & MySQL (mysqli)</title>
04. </head>
05. <body>
06. <form action="save.php" name="frmAdd" method="post">
07. <table width="284" border="1">
08.    <tr>
09.      <th width="120">CustomerID</th>
10.      <td width="238"><input type="text" name="txtCustomerID" size="5"></td>
11.      </tr>
12.    <tr>
13.      <th width="120">Name</th>
14.      <td><input type="text" name="txtName" size="20"></td>
15.      </tr>
16.    <tr>
17.      <th width="120">Email</th>
18.      <td><input type="text" name="txtEmail" size="20"></td>
19.      </tr>
20.    <tr>
21.      <th width="120">CountryCode</th>
22.      <td><input type="text" name="txtCountryCode" size="2"></td>
23.      </tr>
24.    <tr>
25.      <th width="120">Budget</th>
26.      <td><input type="text" name="txtBudget" size="5"></td>
27.      </tr>
28.    <tr>
29.      <th width="120">Used</th>
30.      <td><input type="text" name="txtUsed" size="5"></td>
31.      </tr>
32.    </table>
33.    <input type="submit" name="submit" value="submit">
34. </form>
35. </body>
36. </html>
```

MySQL Stored Procedure in PHP

- save.php

```php
<html>
<head>
<title>ThaiCreate.Com PHP & MySQL (mysqli)</title>
</head>
<body>
<?php
	ini_set('display_errors', 1);
	error_reporting(~0);

	$serverName = "localhost";
	$userName = "root";
	$userPassword = "root";
	$dbName = "mydatabase";

	$conn = mysqli_connect($serverName,$userName,$userPassword,$dbName);

	$strCustomerID = $_POST["txtCustomerID"];
	$strName = $_POST["txtName"];
	$strEmail = $_POST["txtEmail"];
	$strCountryCode = $_POST["txtCountryCode"];
	$strBudget = $_POST["txtBudget"];
	$strUsed = $_POST["txtUsed"];

	$sql = "CALL insertCustomer('$strCustomerID', '$strName', '$strEmail', '$strCountryCode',
	'$strBudget', '$strUsed')";

	$query = mysqli_query($conn,$sql);

	if(!$query) {
		echo mysqli_error($conn);
	}
	else
	{
		echo "Record add successfully";
	}

	mysqli_close($conn);
?>
</body>
</html>
```

Use of Stored Procedure (อ้างอิง)

- SQL Server Stored Procedure in .NET
  - https://msdn.microsoft.com/en-us/library/ms131094.aspx
- SQL Server Stored Procedure in JAVA
  - http://www.thaicreate.com/tutorial/sqlserver-stored-procedure-java.html
- MySQL Stored Procedure in JAVA
  - https://docs.oracle.com/javase/tutorial/jdbc/basics/storedprocedures.html

แนวทางการทดสอบ SQL Injection

- Testing for SQL Injection
  - https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005)
  - https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents
  - http://resources.infosecinstitute.com/best-free-and-open-source-sql-injection-tools/#gref

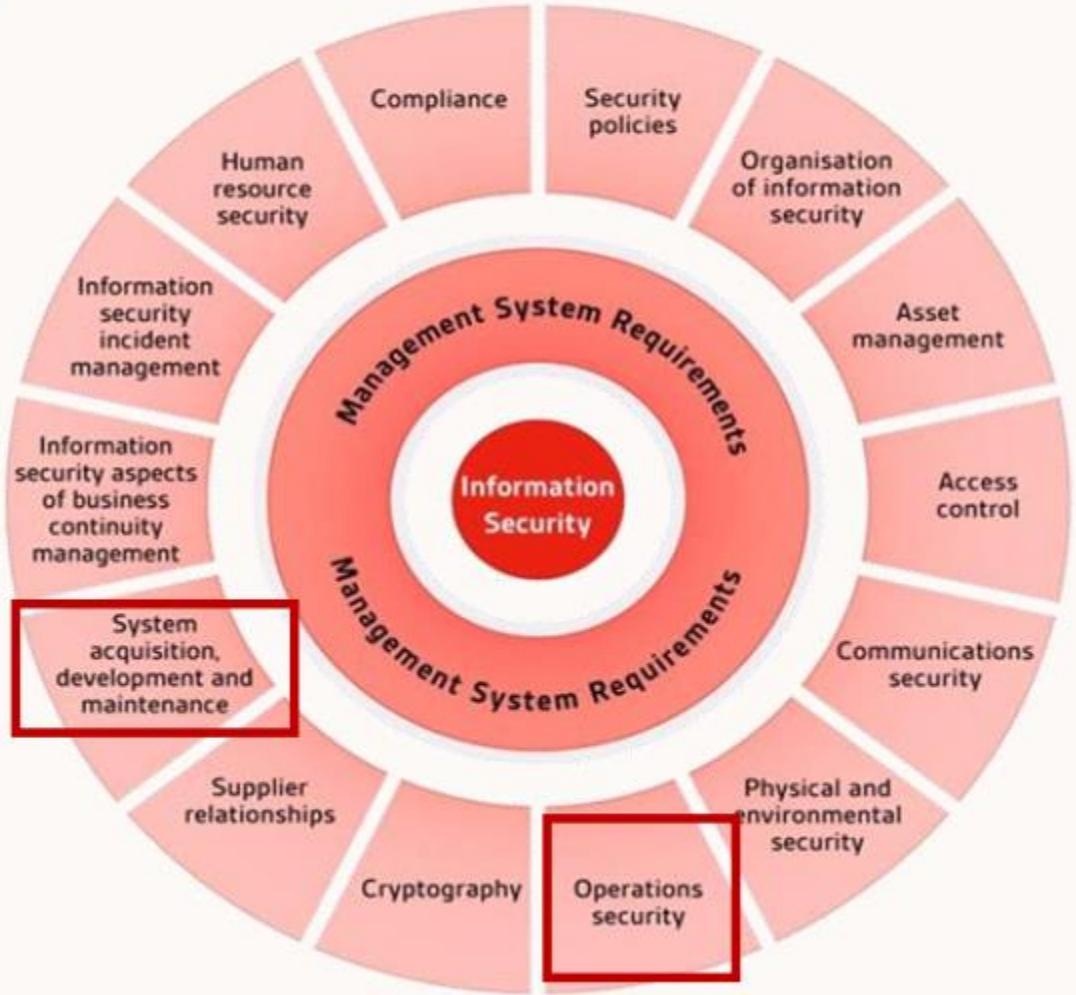
NIST Cybersecurity Framework

Identify	Protect	Detect	Respond	Recovery
การจัดเตรียมสภาพแวดล้อมพื้นฐานเพื่อให้บริการสำคัญมีความมั่นคงปลอดภัย	การป้องกันไม่ให้มีการละเมิดความมั่นคงปลอดภัย	หากมีการละเมิดความมั่นคงปลอดภัย จะมีการตรวจหาว่ามีเหตุการณ์หนึ่งเกิดขึ้นแล้ว	การรับมือกับเหตุการณ์ที่เกิดขึ้น	กรณีที่จำเป็นอาจจะต้องดำเนินการกู้คืนเพื่อกลับไปสู่สภาวะตามปกติ

Framework Core

การเตรียมการ (Identify)

- IT Compliance (Security Domains – ISO/IEC 27001:2013)



14 Domains

114 Controls

A.12.6.1 Management of technical vulnerabilities

- A.12.6.1 การบริหารจัดการช่องโหว่ทางเทคนิค (Management of technical vulnerabilities)
- ความมุ่งหมายของมาตรการ
- ต้องการให้ ติดตาม เฝ้าระวัง ปรับปรุง ข้อมูลเกี่ยวกับช่องโหว่ทางเทคนิคของระบบที่ใช้งาน จุดอ่อนต่อช่องโหว่ดังกล่าวขององค์กรต้องมีการประเมิน และมาตรการที่เหมาะสมต้องถูกนำมาใช้เพื่อจัดการกับความเสี่ยงที่เกี่ยวข้อง

การเฝ้าระวัง (DETECT)

- ManageEngine Vulnerability Manager Plus
- BeyondTrust Retina Network Security Scanner
- Rapid7 Nexpose
- Tripwire IP360
- ImmuniWeb
- Netsparker
- Acunetix
- SolarWinds
- Intruder
- NESSUS
- OpenVAS
- NMAP
- Retina network security scanner
- Nikto

Tools for Vulnerability Scanning

การเฝ้าระวัง (DETECT)

- Technology: SIEM

Source: Gartner (February 2020)

SIEM Pricing

Splunk

ArcSight

McAfee Enterprise Security Manager (McAfee ESM)

RSA NetWitness Logs and Packets (RSA SIEM)

กำหมดราคาตาม ขนาดของข้อมูลที่ส่งเข้า SIEM (GB/s) หรือ (EPS) Event Per Second

IBM QRadar

SolarWinds LEM

กำหนดราคาโดยคิดค่าบริการเป็นแบบสมาชิก รายเดือน หรือ รายปี หรือคิดค่าบริการตามอุปกรณ์ หรือ server

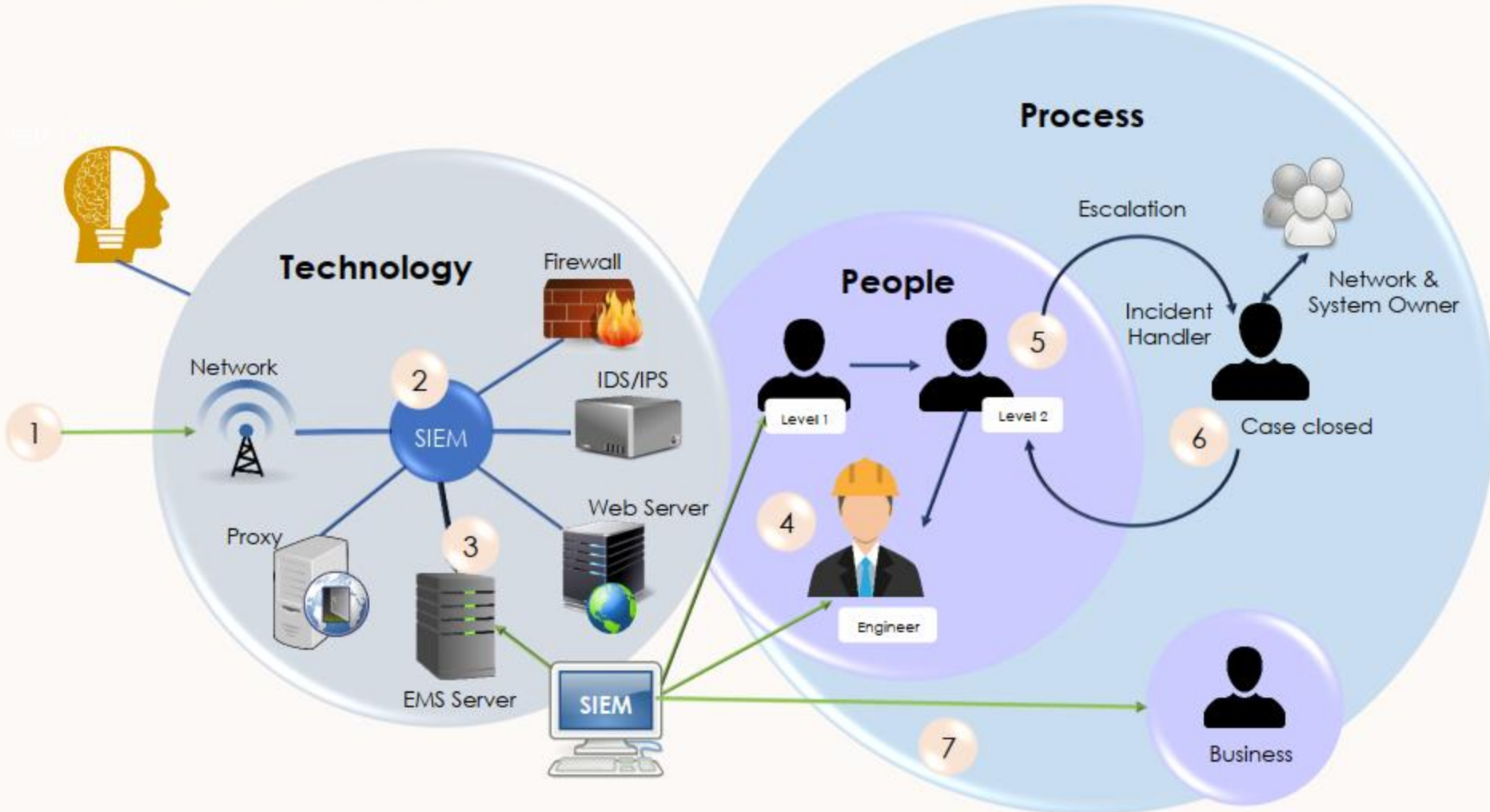
AlienVault

LogRhythm NextGen SIEM

บาง Product เป็น Open Source ต้องสมัครสามาชิก เพื่อใช้งานฟรี แต่มีการเรียกเก็บค่าบริการจาการ Support

การรับมือ (RESPOND)

- SOC Solution Architecture & Design



การกู้คืน (RECOVER)

1. **Veeam Backup & Replication**

8.5	19	336	65,832	51,129
Average Rating	Reviews	Words/Review	Views	Comparisons

2. **Rubrik**

9.5	24	695	47,992	31,550
Average Rating	Reviews	Words/Review	Views	Comparisons

3. **Cohesity**

9.6	20	303	31,161	21,079
Average Rating	Reviews	Words/Review	Views	Comparisons

Q&A

Thank You

Nattawut Opasieamlikit (Head of CSIRT)
0902405079 Nattawut@bcg-ecop.net