國立交通大學

資訊科學與工程研究所

碩 士 論 文

跨平台 Web 程式測試與攻擊產生系統

A Generic Web Application Testing and Attack Generation Framework

研 究 生：劉歡

指導教授：黃世昆　教授

中 華 民 國 一 百 零 二 年 六 月

A Generic Web Application Testing and Attack Generation Framework

研 究 生：劉　歡　　Student：Huan Liu

指導教授：黃世昆　　Advisor：Dr. Shih-Kun Huang

國 立 交 通 大 學
資訊科學與工程研究所
碩 士 論 文

A Thesis
Submitted to College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master of Science
In

Computer Science

June 2013

Hsinchu, Taiwan, Republic of China

中華民國一百零二年六月

跨平台 Web 程式測試與攻擊產生系統

學生: 劉　歡　　指導教授: 黃世昆

國立交通大學　資訊科學與工程研究所　碩士班

摘要

本論文以滲透測試的角度，提出一跨平台網頁應用程式測試與攻擊系統，此系統能針對目標 web 應用程式自動產生攻擊字串，達成滲透測試的效果。

此系統透過網頁爬蟲取得待測 URL，並在 HTTP 要求中插入符號變數以記錄執行過程中的路徑限制式，藉此對現有的網頁應用程式進行脅迫產生。採用方法架構於 S^2E 上，是以 QEMU 為基礎的符號執行環境。由於符號執行的執行時間呈指數成長，為了增進此系統的效率，採用單一路徑符號執行方式來取得路徑限制式。

目前已測試幾種開放原始碼的應用程式，能成功產生相對應的攻擊字串。

A Generic Web Application Testing and Attack Generation Framework

Student: Huan Liu Advisor：Dr. Shih-Kun Huang

Institute of Computer Science and Engineering

National Chiao Tung University

ABSTRACT

This thesis proposed a generic web application testing and attack generation framework. This system can automatically generate attack strings for the target system, just like penetration test.

This system uses a web crawler to explore URLs, and generate HTTP requests. Each test sends symbolic variable to the target server in order to record path constraint. It can solve constraints of exploit from those gathered paths. This system is based on S^2E, a symbolic environment based on QEMU. In order to improve efficiency of symbolic execution, this system uses single path concolic execution to generate web application exploit.

Finally, we have applied this system to several known vulnerabilities on open source web applications, and generated the corresponding exploit successfully.

誌謝

首先感謝我的指導教授黃世昆老師。老師讓我可以自由地探索軟體安全這個領域，也耐心地指導我解決研究過程中所碰上的問題，同時也常常關心實驗室的學生們，讓軟體品質實驗室的人們感到非常溫馨。

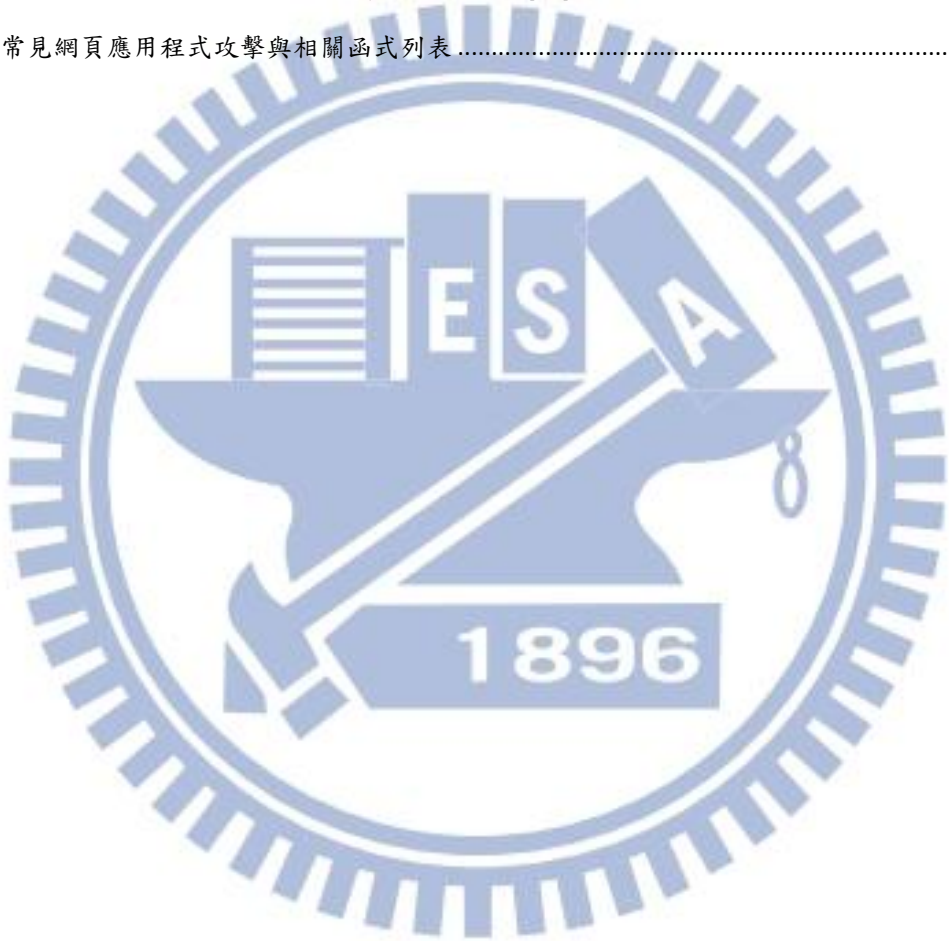
接著要感謝我的父母，沒有你們就沒有現在的我。我相當幸運的能夠擁有這樣一個完整而美好的家庭。

再來，要感謝偉明，沒有你之前的成果就沒有這篇論文。所以還要追溯感謝博彥學長。感謝銘祥學長帶我學習 S2E 的基礎，還有肇鈞學長對此篇論文的建議，讓它變得更加完整。感謝基傑和俊維還有韋翔給實驗室帶來快樂的生活。

感謝 Jeff 和 linpc 還有介豪，在這三個月寫論文的路上一起奮鬥，謝謝你們。感謝芊慧幫忙看論文修投影片，感謝花太太每天忍受我永無止盡的牢騷。

目錄

表目錄



圖目錄

第一章　緒論

1-1　研究動機

在資訊科技發達的現今，人們變得日益依賴網際網路，其已成為生活中不可或缺的一部份。隨著網路普及度提升，每個人在網路上所能夠存取的資料量變得更大，範圍更廣，舉凡金融交易、個人私密資料等皆可透過網頁存取，也因此網頁安全性亦隨之變得越發重要。

然而網頁開發者在開發的途中可能由於各種原因而忽略了安全的考量，導致駭客透過開發者不慎留下的漏洞進行資料的非法存取或破壞。為了減少這類的問題，網頁安全的領域已提出各種不同的方法來檢測漏洞的存在及可利用性。

在此以攻擊者的角度，探討自動產生網頁攻擊的方法。此法可應用於檢測網頁中是否存在相關漏洞，協助網頁管理者能夠盡速修復。

1-2　研究目標

本論文的目標是能夠自動產生網頁應用程式的脅迫 (Exploit)，相當於一般的手動滲透測試。本研究是使用符號執行 (Symbolic Execution [1, 2]) 為基礎，對受測網頁應用程式進行動態分析。

在此使用本實驗室開發的 CRAX[3] 工具，這是一個使用單一路徑擬真執行 (Single Path Concolic Execution) 來進行跨平台安全性分析的工具。基於 CRAX 可以跨平台且涵蓋不同程式語言，使測試上不受系統或語言的限制；加上能夠支援 Symbolic socket，故本篇論文使用為測試工具。

本篇論文將討論利用 CRAX 自動產生網頁應用程式脅迫的方法，提出一方便佈署應用程式的測試框架 (framework)。

1-3　論文大綱

本論文共分為六個章節：

第一章 緒論，說明研究動機與研究目標。

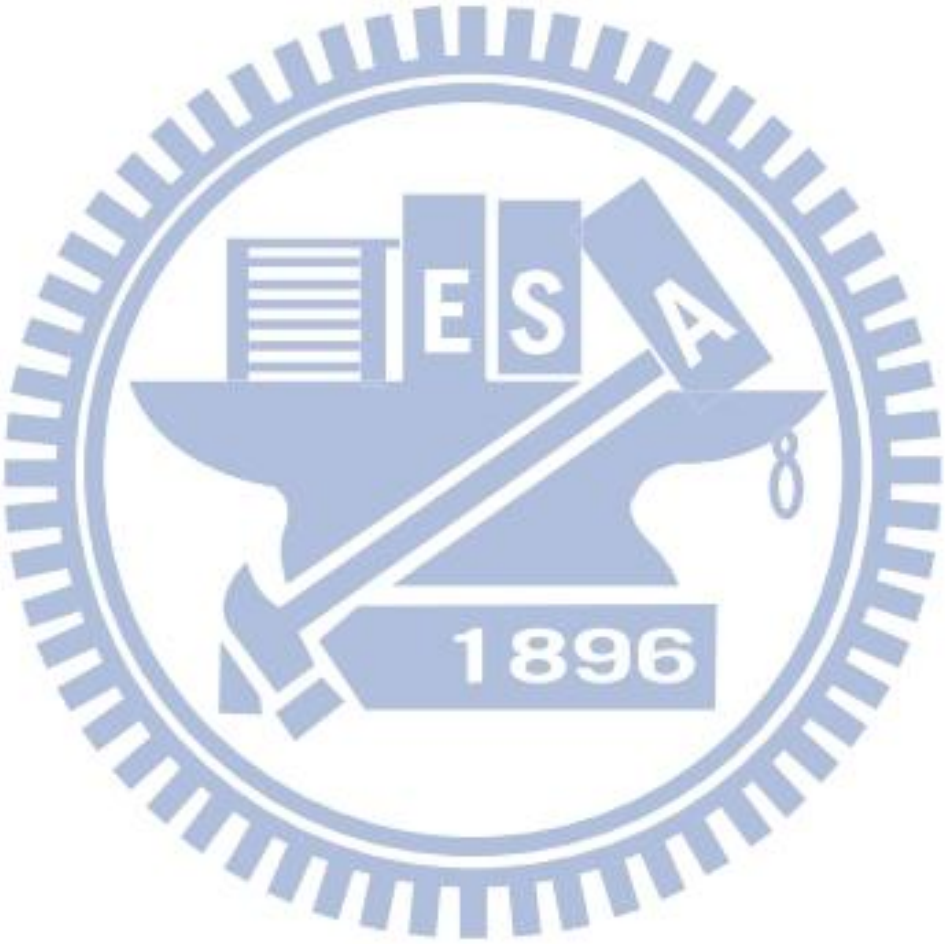
第二章 研究背景，介紹與論文研究相關的知識，包括軟體測試與網頁安全背景知識，並比較近五年來幾種針對網頁應用程式的安全測試系

統。

第三章 研究方法，說明符號執行的環境、系統架構及測試流程。

第四章 實驗結果與分析，與 MIT 所開發的 Adrilla 比較，並展示實際測試的結果。

第五章 結論與未來展望，為本論文的結論與未來研究方向。



第二章　研究背景

2-1　軟體品質測試

軟體測試是在軟體開發過程中不可或缺的一環，其主要目的就是提早發現錯誤，以增加軟體的可靠度和正確性。

2-1-1　符號執行

符號執行 (Symbolic Execution) 是一種常見的動態分析，它的目標是用來測試程式的路徑覆蓋率 (path coverage)。符號執行會將受測程式的資料流用符號變數 (symbolic variable) 來取代，其可視為一值域。

在符號執行中，當一般變數取用了符號變數的值，那麼一般變數便會成為一符號變數。

若該變數進入分支條件 (branch condition) 時會分別產生兩組對應的限制式 (constraint)，一組滿足分支條件，使其為真；另一組則使其為非。這兩組限制式被稱為路徑限制式 (path constraint)，分別代表不同的程式執行路徑。當符號變數再次進入分支條件時會產生新的限制式，若新的限制式能夠滿足原本的路徑限制式，則與原本的路徑限制式合併而成新的路徑限制式；若無法滿足則丟棄，代表此路徑已走到終端。

符號執行完畢後所得到的路徑限制式代表唯一一條程式執行路徑，且可透過 solver 產生一組執行路徑相同的測資 (test case)。

考慮以下程式碼 (圖 1)：將第一行的變數 x 取代為符號變數 X，在執行到第四行時進入分支條件。此時程式會分成兩種不同的狀態繼續執行，此兩種狀態分別為 X >=0 和 X<0，同時會產生兩組對應的限制式，如圖 2 所示。

```
01.  void foo (int x)
02.  {
03.      int y = 0;
04.      if (x >= 0)
05.          printf("state 1");
06.      else
07.          printf("state 2");
08.
09.      y = x + 100;
10.
11.      if (y >= 0)
12.          printf("state 3");
13.      else
14.          printf("state 4");
15.  }
```

圖 1 範例程式碼

程式繼續執行，至 11 行時會再次碰到分枝，此時程式會分成四種狀態，且產生四組路徑： (X>=0) ^ (X+100>=0)、(X>=0) ^ (X+100<0)、(X<0) ^ (X+100>=0)、(X<0) ^ (X+100<0)。這四組路徑中，(X>=0) ^ (X+100<0) 是一組矛盾的限制式，因此會被符號執行捨棄。

程式執行完畢後，符號執行總共產生了三條限制式，分別代表不同的執行路徑。詳細流程見圖 2。

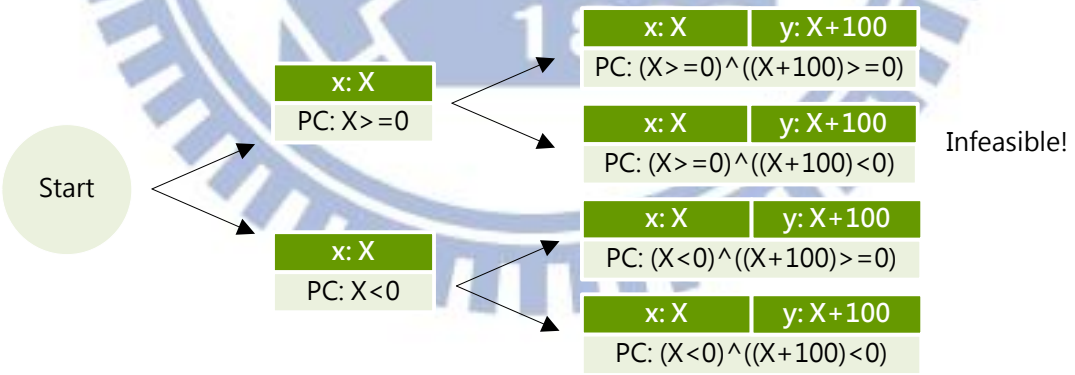


圖 2 圖 1 之程式碼使用符號執行流程圖

2-1-2 擬真執行

符號執行在某些狀況會有無窮迴圈的問題，例如 GUI 程式會利用無窮

迴圈來偵測事件的發生。且符號執行的時間複雜度呈指數成長，當受測程式規模變大的時候會相當沒有效率。為了解決這個問題，擬真執行 (Concolic execution [4]) 的構想被提出。

擬真執行是一種路徑選擇的方法，其包含具體執行 (concrete execution) 以及符號執行的特性。擬真執行每次只會探索一條執行路徑，其以一組隨機的輸入作為初始直開始執行，和符號執行相同，在執行過程中擬真執行會將執行過程紀錄於路徑限制式中。

執行完畢後，其根據否定的路徑限制式裡的分支限制式 (branch constraint) 產生新的測試資料，直到全部的路徑被執行完畢為止。

考慮圖 1 的程式碼，首先產生一個隨機值來初始化 x，假設這個隨機值為 3 (見圖 3 步驟 1)，經過擬真執行之後會得到路徑限制式：(X>=0) ^((X+100)>=0)。為了探索所有的執行路徑，擬真執行取適才得到的路徑限制式的否定限制式加以探索 (圖 3 步驟 2)，發現這個路徑限制式矛盾，捨棄之。

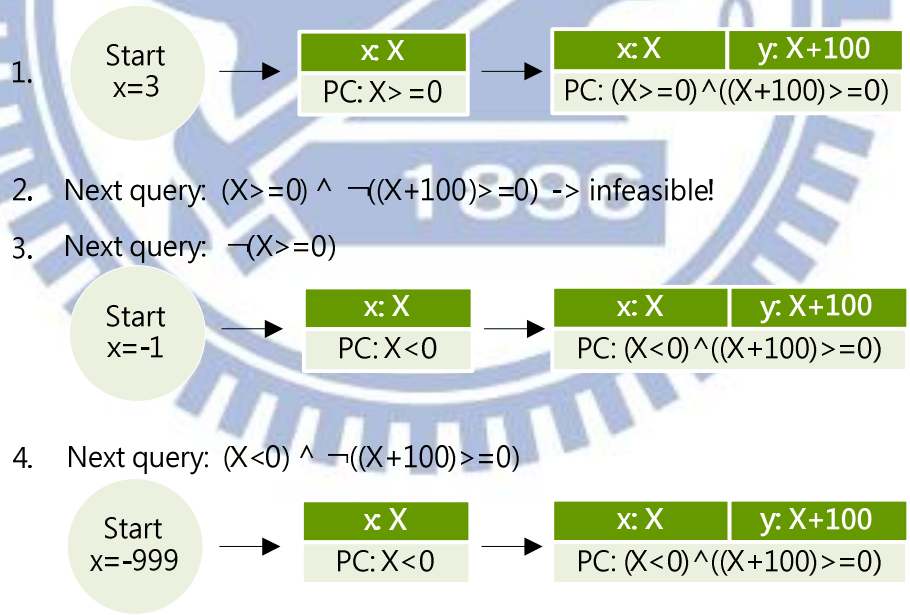


圖 3 圖 1 之程式碼使用擬真執行流程圖

接著依序探索剩下的路徑限制式之分支，以 (X<0) 為限制產生亂數，在此設 x 值為 -1 ，得路徑限制式 (X<0)^((X+100) >=0)。後繼續依此規則進行路徑探索，詳細流程見圖 3。

2-1-3　　單一路徑擬真執行

單一路徑擬真執行的目的不在於檢測路徑覆蓋率，而是注重在蒐集執行時所經過的控制流程 (control-flow) 資訊。單一路徑擬真執行以給定的特定輸入值開始執行，將執行途中的分枝資訊紀錄於路徑限制式中，直到程式執行完畢為止。由於不進行路徑探索，單一路徑擬真執行只會執行一次。

考慮圖 1 的程式碼，假設 x 的初始值為 1，在第四行時遇到第一個路徑分枝，紀錄路徑限制式 X>=0 之後繼續往下執行；於第 11 行時碰到第二個路徑分枝，改變路徑限制式：(X>=0) ^ (X+100>= 0)，之後程式結束執行 (圖 4)。

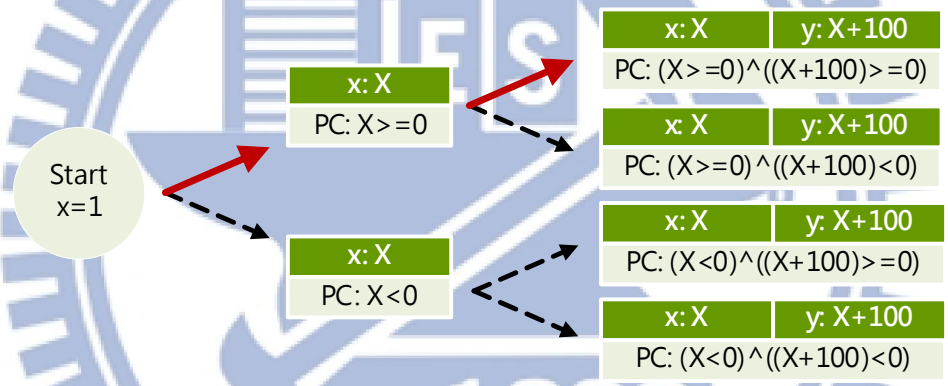
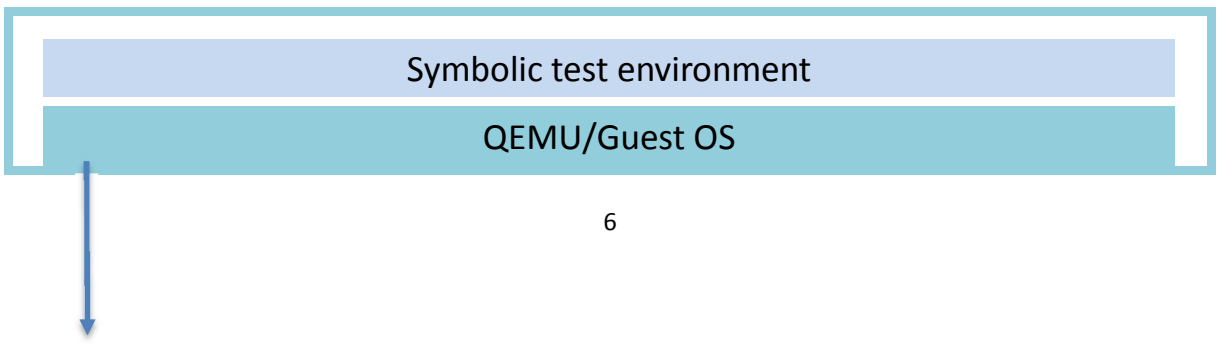


圖 4 圖 1 之程式碼使用單一路徑擬真執行流程圖

2-1-4　　基於 S²E 的符號環境

S²E [5] 是一個分析軟體行為的平台，其建置在 QEMU [6] 之上，以 KLEE [7] 為符號執行引擎，提供全系統的符號執行及擬真執行的測試環境。

由於建置在 QEMU 之上，S²E 可以方便地做真實環境的測試，如函式庫 (library)，系統核心 (kernel)，或底層驅動程式 (drivers) 皆可測試。



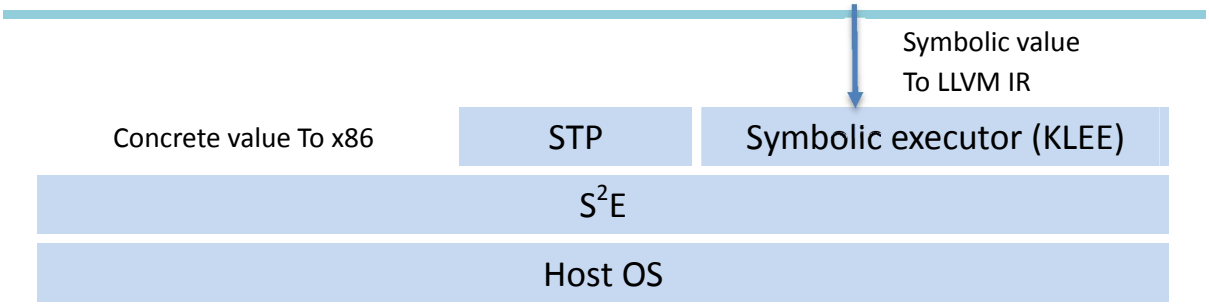


圖 5 S2E 架構圖

2-2 網路安全

本篇論文是為了應用於偵測網頁應用程式中可能的漏洞，以下章節會介紹近年來較為重大、受害程度較廣的幾種網頁應用程式的安全弱點。

2-2-1 OWASP TOP 10

開放網路軟體安全計畫 (The Open Web Application Security Project, OWASP) 是一個開放社群，其主要目標是協議並解決網路軟體安全之標準、工具及技術文件，並協助政府及企業改善應用程式的安全性。

開放網路軟體安全計畫每年會提出前十大網站安全弱點(OWASP TOP 10[8])，提供開發者基本方法來防止這些弱點，以提升軟體安全性。此十大安全弱點並非依常見程度排序，而是包含了發生的可能性與其可能造成影響的程度。

2013 年前十大網站安全弱點茲列如下：

A1. 注入攻擊 (Injection)

A2. 遭破壞的鑑別與連線管理 (Broken Authentication and Session Management)

A3. 跨站腳本攻擊 (Cross-Site Scripting)

A4. 不安全的物件參考 (Insecure Direct Object References)

A5. 網站安全組態不當設定 (Security Misconfiguration)

A6. 敏感數據的曝光 (Sensitive Data Exposure)

A7. 函數執行權限的不當控制 (Missing Function Level Access Control)

A8. 跨站冒名請求 (Cross-Site Request Forgery, CSRF)

A9. 使用有已知安全弱點的元件 (Using Components with Known Vulnerabilities)

A10. 未驗證的網頁重新導向 (Unvalidated Redirects and Forwards)

2-2-2 跨站指令碼

跨站指令碼攻擊(Cross-Site Scripting，簡稱 XSS)是一種常見的網頁攻擊方式，位於 OWASP 中網站應用程式十大弱點中。攻擊者透過網頁漏洞，在網頁中植入惡意代碼，其他使用者在觀看網頁實就會受到影響。這類攻擊通常包含 HTML 以及客戶端之腳本語言，例如 JavaScript、ActiveX 等。

跨站指令碼依照攻擊手法的不同，可分為兩種類型。其一為反射性(Reflected)，其二為儲存性 (Stored)。

反射性漏洞的攻擊中，攻擊者會製造一包含可執行代碼的連結送予被害者，當被害者打開連結時，便會執行攻擊者所附上的代碼，此為反射性跨站指令碼漏洞的利用方式。

儲存性漏洞與反射性漏洞相若，差別在於儲存性漏洞中的惡意程式能夠被攻擊者置放於該網站之程式碼或者資料庫內，所有瀏覽過該網頁的使用者皆會受到危害。

以反射性跨站指令碼為例，使用者點選示意連結時 (如圖 6 左側瀏覽器所示)，會顯示 'Hello, World!'。由於此擷取網址參數並顯示於頁面中的程式並未過濾使用者輸入，因此攻擊者可利用此漏洞令使用者在點選連結時執行攻擊者所包含於其中的惡意程式 (見圖 6 右側瀏覽器示意圖)。

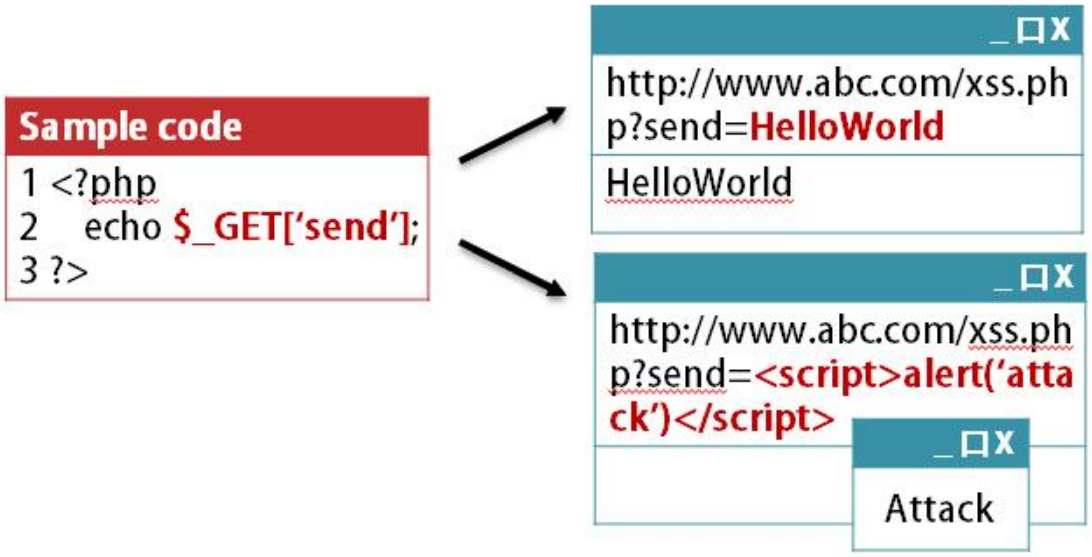


圖 6 跨站指令碼攻擊示意圖

2-2-3 注入攻擊 (Injection)

注入攻擊 (Injection) 同樣為一常見之網站應用程式攻擊手法，位列於網路應用程式安全計畫中網站應用程式十大弱點之一。常見有 SQL 資料隱碼攻擊 (SQL Injection)、LDAP 注入攻擊 (LDAP Injection)、指令注入攻擊 (Command Injection) 等。

此類攻擊手法利用網頁中忽略檢查輸入字串合法性的漏洞攻擊，以 SQL 資料隱碼攻擊為例，其在正常的資料庫指令中夾帶攻擊者所下的惡意指令，使伺服器認為該指令為正常行為而執行，因而使資料遭到竊取或破壞。

指令注入攻擊略有不同，其發生於網頁應用程式直接呼叫系統上的其他程式，如 system()，當此類函式呼叫沒有檢查輸入字串的合法性時，便會造成指令注入攻擊。

以圖 7 為例，此為一網頁應用程式的登入畫面，該程式會向資料庫送出查詢確認使用者輸入的帳號與密碼是否符合。然而若此程式未檢查輸入合法性，如圖所示，在 # 之後的字串會被視為註解，此查詢式永遠為真，攻擊者便可藉由輸入惡意指令來冒充使用者登入竊取資料。

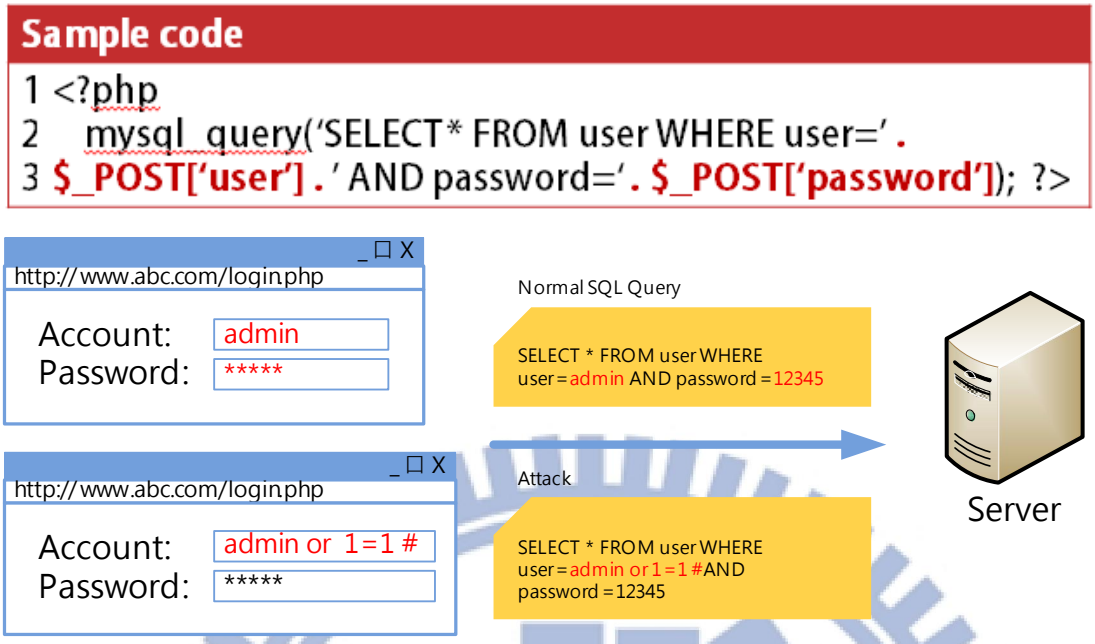


圖 7 SQL 資料隱碼攻擊示意圖

2-3 自動產生 web 程式攻擊檢測系統

在以下章節將會介紹數種檢測網頁應用程式弱點的系統，包含自動產生網頁應用程式之脅迫，及偵測出網頁應用程式之漏洞的系統。在此粗略將這些系統分作兩類：其一為使用符號執行引擎，以及使用其他方式的系統。

2-3-4 使用符號執行引擎進行檢測

在此小節將會討論數種與 CRAX 相同，近年來使用符號執行作為測試主軸的網頁程式攻擊/弱點檢測系統。

SAFELI[9] 於 2008 年提出，利用分析網頁應用程式的原始碼來產生攻擊字串，是一較早期針對 JAVA 網頁應用程式的資料隱碼攻擊產生器。同年 MIT 提出了 Apollo[10] ，其更改 PHP 的原始碼令其支援擬真執行，用以找出 PHP 網頁應用程式中的漏洞。

隔年 MIT 改進 Apollo ，提升其功能，此專案名為 Adrilla[11]，為一針對網頁應用程式的自動化脅迫產生系統。其統合擬真執行與動態分析來產生脅迫，是一個針對 PHP 平台的系統。Adrilla 並針對其中幾個開放原始碼的 PHP 網頁應用程式做了弱點分析，在本篇論文稍後部分會對相同的應用程

式進行與 Adrilla 的效能比較。

Kudzu[12] 於 2010 年提出，是一針對 JavaScript 的框架。其使用攻擊文法作為脅迫產生的基礎，此工具已成功發現兩個未知的弱點。

2012 年 Huang, Chen et al. [13]針對 JAVA 網頁應用程式的弱點分析提出了和 Adrilla 相似的概念，略有不同的是此篇論文使用動態汙點分析技術 (dynamic taint analysis) 來產生所有的程式執行路徑，之後再使用符號執行獲得路徑限制式供弱點分析使用。

這類方法能夠完整的分析受測程式，然而需要受測程式的原始碼進行靜態分析，並且侷限於單一程式語言。

2-3-5 使用其他方式進行檢測

在此小節將會討論近年來不使用符號執行，而使用其他方式作為檢測/攻擊產生引擎的網頁程式攻擊/弱點檢測系統。

PIUIVT[14] 於 2010 年提出，此工具會分析網頁中的文字來辨識輸入的格式，進而提高產生攻擊字串的效率。其根據分析測試目標的回傳值來修正攻擊字串，進而達到成功攻擊目標的效果。2011 年，Bashah Mat Ali 等提出了 MySQLInjector[15]，其使用 Blind SQL Injection ，同樣是以網頁應用程式的回傳值為攻擊成功於否的基準。

於 2012 年，Tian 等提出 NKSI Scan[16]，與 Blind SQL Injection 相若，同樣是以網頁應用程式回傳值加以判定攻擊是否成功。其將滲透測試中使用的字串進行模組化，加以排列而成測試資料。

這類方式屬於完全黑箱測試，不需要待測目標的原始碼即可進行弱點分析；然而此類測試只適用於漏洞屬於未加密的 SQL 指令，對於以加密保護的應用程式能力較弱。

第三章　研究方法

此一章節將闡述 CRAX 所實現的網頁應用程式的測試方法及流程。CRAX 是基於 S^2E 的符號執行環境所建置，在此環境下使用 socket 來傳送、接收資料，利用 socket 送出符號資料 (Symbolic data) 至伺服器端來測試目標網頁應用程式。

在伺服器端，我們更改特定函式來偵測符號資料；於客戶端，我們在接收到伺服器端回傳值後對其進行符號資料的偵測。偵測到符號資料後，呼叫脅迫產生器 (Exploit Generator) 進行分析、產生脅迫。

3-1　系統架構

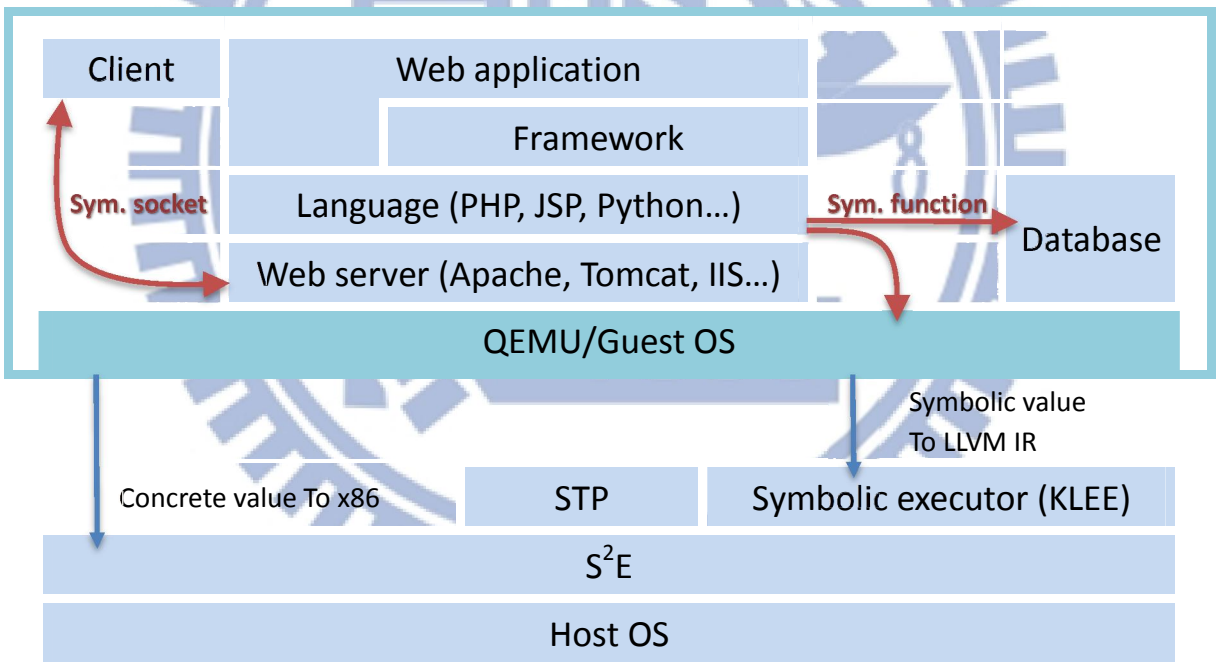


圖 8 CRAX 在 S^2E 上的實驗環境

CRAX 建立在 S^2E 平台之上，於 QEMU (Guest OS) 中部署網頁伺服器與待測應用程式及客戶端 (參見圖 8)。

客戶端與伺服器端使用 Symbolic socket 相互溝通，網頁伺服器端與資料庫、作業系統間互動的函式中加入符號資料偵測器 (Symbolic data detector)

以偵測符號資料，最後將符號資料送至脅迫產生器 (Exploit generator) 產生脅迫。

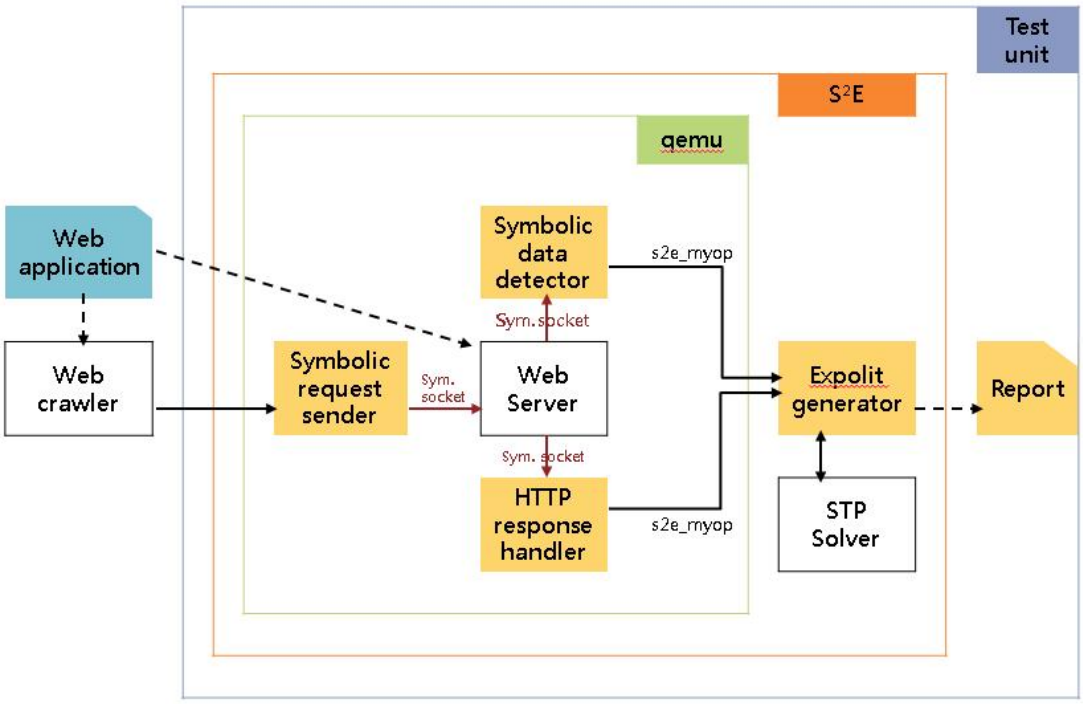


圖 9 CRAX 架構

為了方便實驗報告統整及實驗資料儲存，CRAX 使用一網頁作為前端介面。使用者在前端選擇待測應用程式，控制節點 (Control Node) 便會初始化運算節點 (Computing Node)。

運算節點上為 CRAX 系統 (如圖 9)，CRAX 接收控制節點所發出的待測 HTTP 要求，開始進行弱點分析。分析完的報告會上傳至資料庫，並在前端顯示實驗結果。

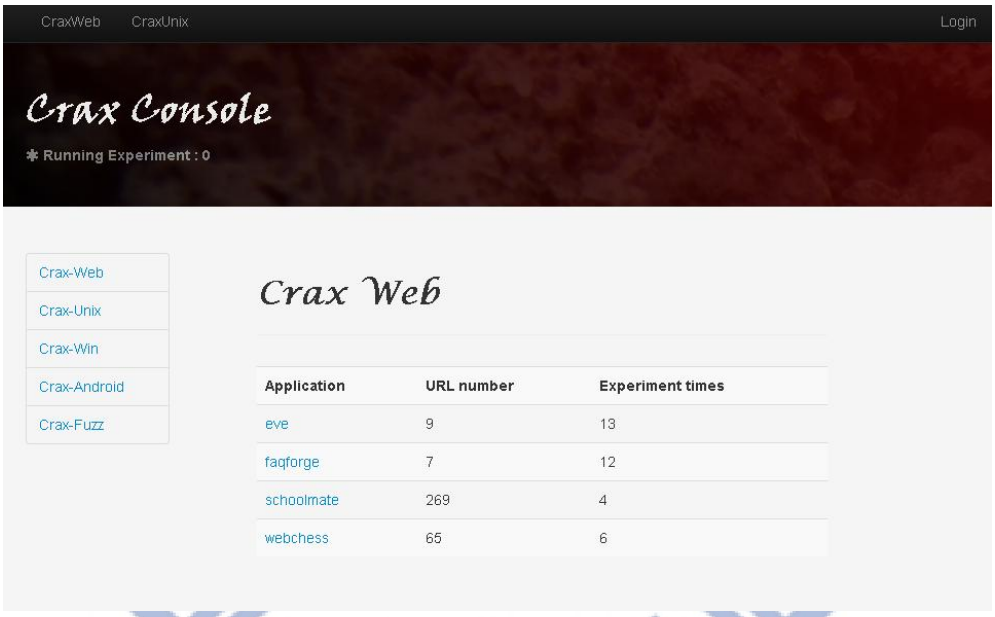


圖 10 CRAX 網頁介面

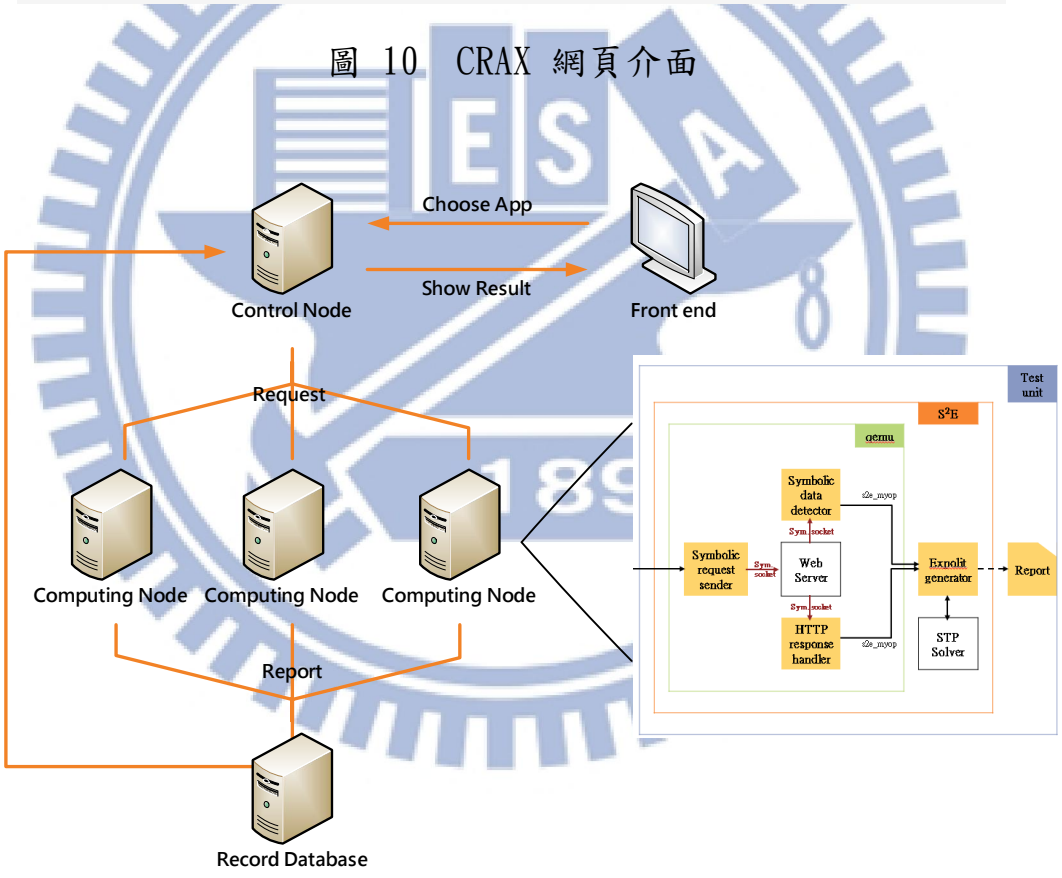


圖 11 CRAX 實驗環境

3-2 Web crawler

Web 程式取得使用者輸入資料的方式有二：GET 和 POST 指令[17] 。惡意攻擊者會透過以上兩種方式對伺服器送出可疑要求，從而進行攻擊。

網頁爬蟲 (Web crawler) 會取得該網頁應用程式的所有頁面，分析其中的 GET 和 POST 指令。它會找出並標示這些 HTTP 要求的參數，將這些要求和參數作為 CRAX 的輸入資料。

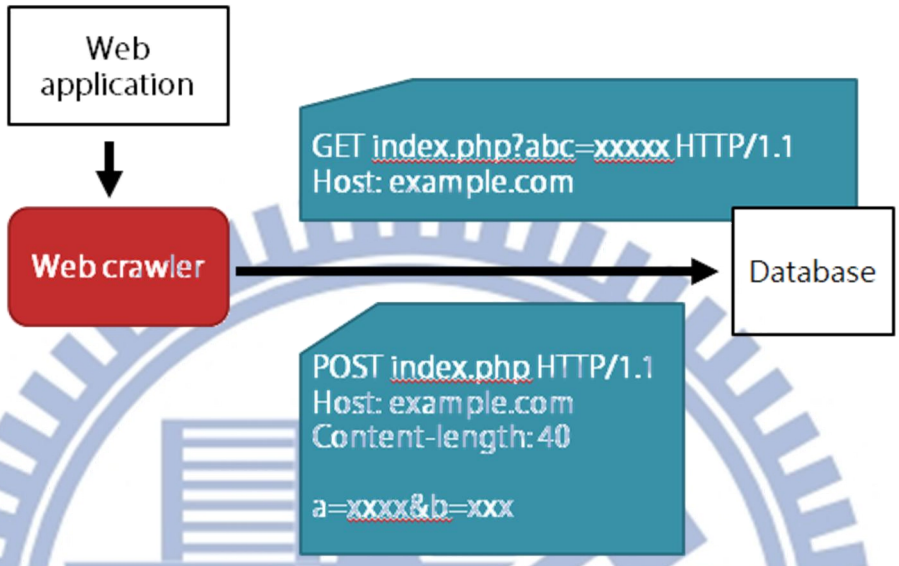


圖 12 網頁爬蟲的運作方式

3-3 符號資料發送器及偵測器

符號資料發送器會取得網頁爬蟲所送出的資料，將取得的 HTTP 要求部分用符號資料取代之後送出至伺服器端 [18]。若是這些符號資料被伺服器端處理過後能夠抵達資料庫進行查詢；或者對作業系統進行變更；又或經由 HTTP 回覆傳至客戶端等，則代表以上這些行為能夠經由符號資料來操縱。

為了偵測伺服器端的符號資料，我們針對常見的攻擊行為，在一些敏感函式中佈署符號資料偵測器(Symbolic data sensor)。以 PHP 為例，SQL 資料隱碼攻擊是對資料庫進行不合法的查詢，可佈署於 mysql_query() ；遠端文件包含攻擊可佈署於 include() 、 require() 等。在這些位置偵測符號資料，以確認這些函式的參數是否可被輸入所操縱 (流程圖詳參圖 13)。

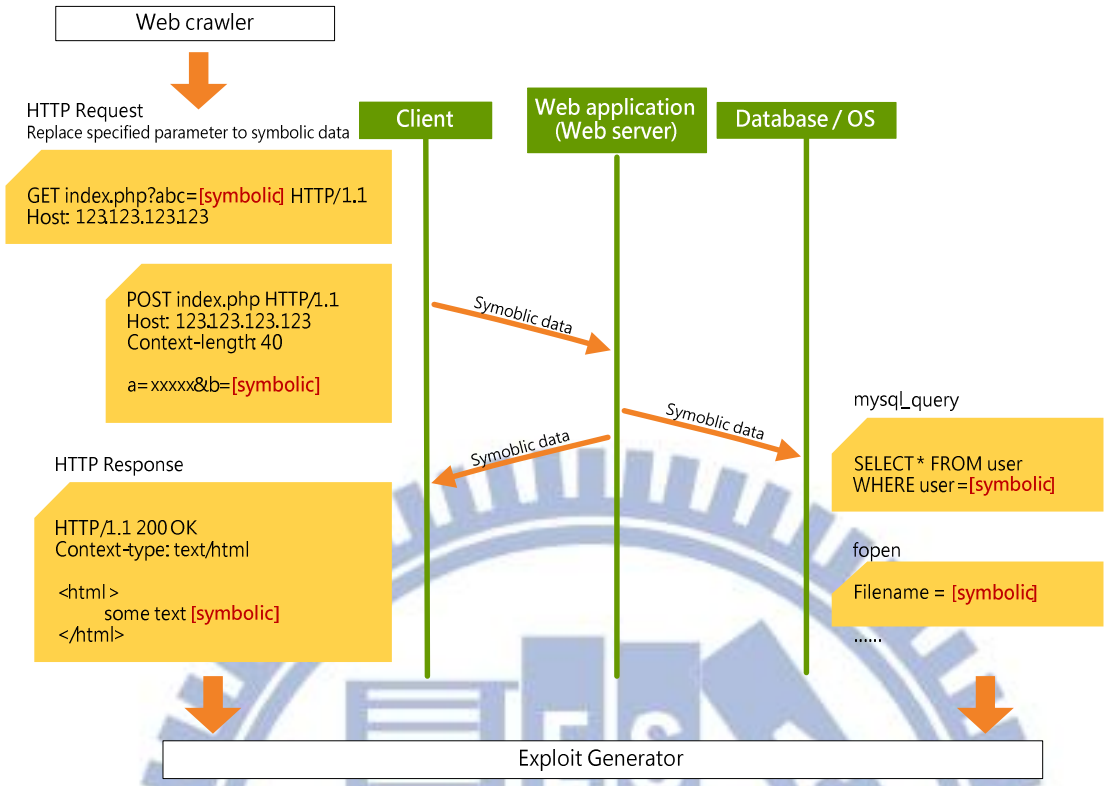


圖 13 符號資料發送器及偵測器運作示意圖

在實作上，符號資料發送器向控制節點發送實驗要求，控制節點則從資料庫中取出網頁爬蟲所分析出的 HTTP 要求和需要取代成符號資料的位置後將其送給符號資料發送器。這種方式可以根據實驗要求，彈性變更需要取代成符號資料的位置。符號資料發送器會與伺服器端建立連線，並建立一塊符號記憶體，將該記憶體內容作為資料送往伺服器端。流程見圖 14。

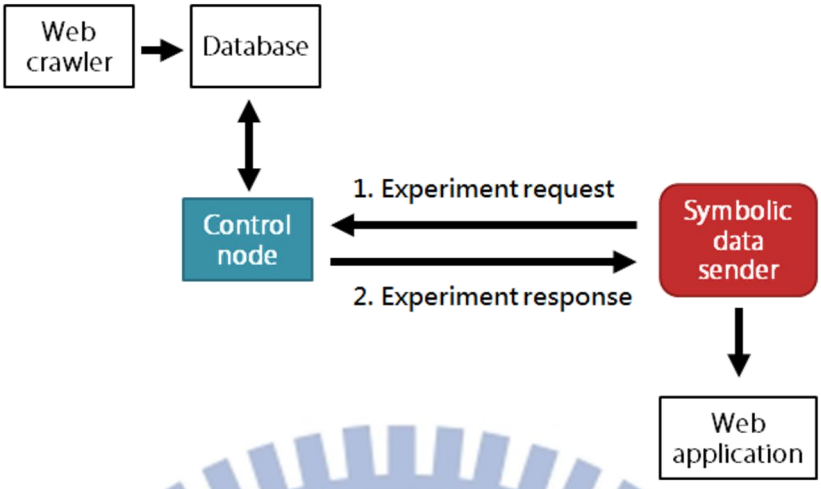


圖 14 符號資料發送器運作流程

符號資料偵測器則是透過客製化的 S^2E 指令來時作，在此稱之為 s2e_myop。這個指令會被符號資料偵測器所呼叫，符號資料偵測器透過 s2e_myop 將需要分析的字串傳至 Host 處理。

3-4 脅迫產生器

脅迫產生器的原理是選定可能的目標攻擊指令，以此為依據產生對應的脅迫碼。

3-4-1 限制式求解

在 2-1-1 中有提到，符號執行可以透過路徑限制式來產生具有相同執行路徑的測試資料。在此節，我們將利用此一性質來進行脅迫的產生。

如圖 15 所示，若已知程式輸出為 y 以及此程式執行過程中所收集的路徑限制式，欲求得未知輸入 x。限制式求解可將 y 的值作為限制，與路徑限制式合併求解，即可得出經過此執行路徑，產生輸出值為 y 的輸入值 x 。

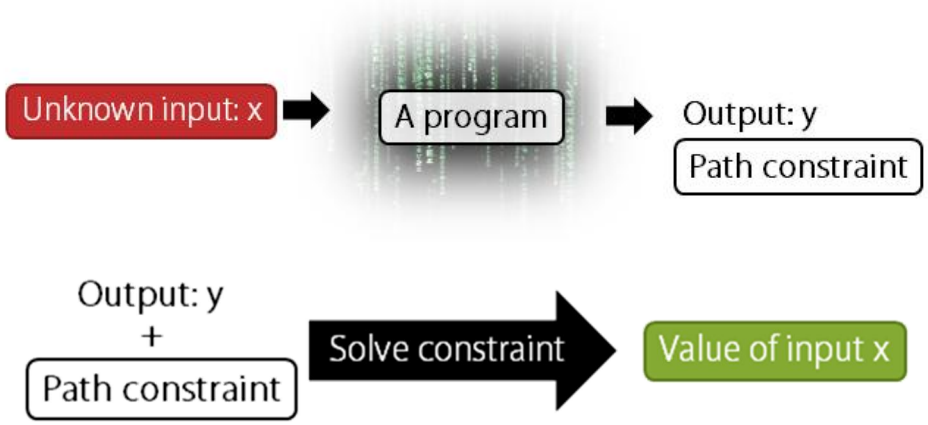


圖 15 限制式求解原理

考慮圖 16 之程式碼，若已知 bar(x) 的值為 100，可以利用 solver 來求 x 的值。利用單一路徑擬真執行收集 x 的執行路徑，可得 bar(x)=100 的路徑限制式為 x+100>=0，再加上回傳值 y=100 這兩個條件下去求解，可得運算式 x+100＝100。使用 solver 求解此算式，可得解 x=0。

圖 16 限制式求解範例程式碼

此一性質同樣適用於產生脅迫上，在此以 SQL 資料隱碼攻擊為例。考慮圖 17 ，於圖中我們已知伺服器會送出該資料庫查詢式，產生脅迫目標為令該資料庫查詢式夾帶攻擊指令。假設攻擊指令為 admin or 1=1-- ，此指令會使該資料庫查詢式回傳值永遠為真。由於此查詢式中含有符號資料，其中帶有執行過程中的路徑限制式，solver 可以透過此路徑限制式和預期的攻擊指令來解出脅迫碼。

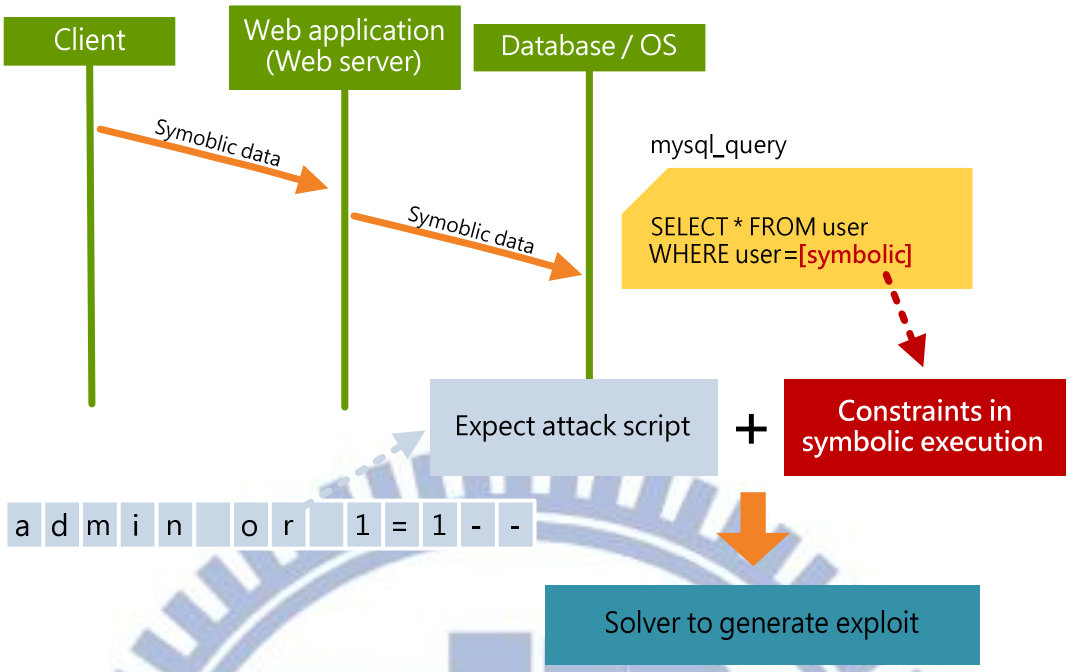


圖 17 脅迫產生器運作流程

在實作上，所有經過符號資料偵測器的資料均會送至脅迫產生器檢驗是否含有符號資料。由於符號資料偵測器只會散佈於敏感函式之中，亦即，此操作可檢驗符號資料是否會到達這些敏感函式。

圖 18 符號資料偵測器運作流程

若脅迫產生器偵測到符號資料，其便會從資料庫中取出常見的 SQL 資料隱碼攻擊之攻擊指令，和路徑限制式一同交由 solver 嘗試解出脅迫碼。

為了要產生能夠順利運行的攻擊指令，脅迫產生器同時會分析整個查詢式，並據結果來更動攻擊指令。以表 1 列一為例，由資料庫取出的攻擊指令為 admin or 1=1，分析資料庫查詢式後發現符號資料前後無特殊字元，

不需要更動攻擊指令。

又，見表 1 列二，此資料庫查詢式中的符號資料前後由單引號括起，故需由原本的攻擊指令 admin or 1=1 更動為 admin' or 1=1-- 或者 admin' or 1=1 #。

#	SQL Query	Attack Script
1	SELECT * FROM user WHERE user=[symbolic]	admin or 1=1
2	SELECT * FROM user WHERE user='[symbolic]'	admin' or 1=1-- admin' or 1=1#

表 1 產生可用的攻擊指令

3-4-2 脅迫產生器演算法

脅迫產生器的原理是選定可能的目標攻擊指令，以此為依據產生對應的脅迫碼，目標攻擊指令和脅迫碼均為一字串。

在目標攻擊指令的位置處會偵測到一塊符號資料，由於已選定可能的目標攻擊指令，而目標攻擊指令為一字串，脅迫產生器首先會偵測這塊符號資料的長度是否大於目標攻擊指令：若長度比目標攻擊指令小，表示無法產生對應長度的攻擊指令，亦即無法產生對應的脅迫碼。

若符號資料的長度比目標攻擊指令長，則將目標攻擊指令加進路徑限制式中，並呼叫 S^2E 的內建 solver 來試圖解出脅迫碼。(演算法見表 2)

	Exploit Generator Algorithm
1	string exploit_generator(symbolic symbolic_data){
2	if(symbolic_data.length < target_attack.length)
3	return NULL;
4	symbolic_data.add_constraint(target_attack);
5	return solve(symbolic_data);
6	}

表 2 脅迫產生器之演算法

第四章　實驗結果與分析

4-1　實驗環境

在本篇論文的所有實驗都是在圖 9 之架構下執行。

硬體環境部分，計算節點為一佈署於 Vmware Esxi 上的虛擬機器，其作業系統是為 Ubuntu Server 10.04 64-bit，記憶體為 6 GB；Guest 環境是一以 QEMU 作直譯器的虛擬機器，其作業系統為 Debian 5.0.7 32-bit，見表 3。

軟體部分，我們使用 S2E 1.0 來架構符號環境，受測伺服器為 Apache ，資料庫是為 MySQL 5.5.15。

Name	OS	CPU	memory
Computing Node	Ubuntu server 10.04	Intel(R) Xeon(R) CPU E3-1230 V2 1 core	6GB
Guest environment	Debian 5.0.7 32bit	2.83GHz	128MB

表 3 實驗環境之硬體規格

本篇論文中大多數的實驗操作皆由前端網頁介面執行，使用者可由網頁控制台選取待測應用程式並新增實驗 (圖 19)，此頁面可同時觀看實驗進度。接著至網頁監控端開啟實驗 (圖 20)，在此可以直接監控實驗進行流程。

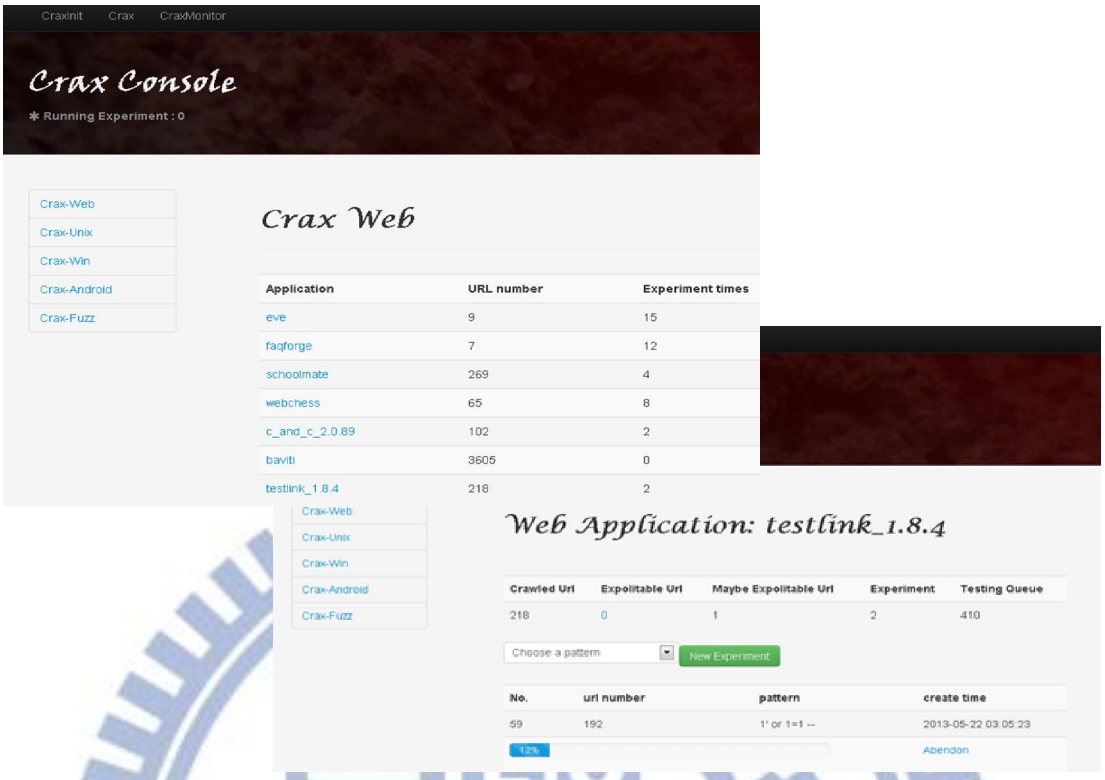


圖 19 CRAX 網頁介面

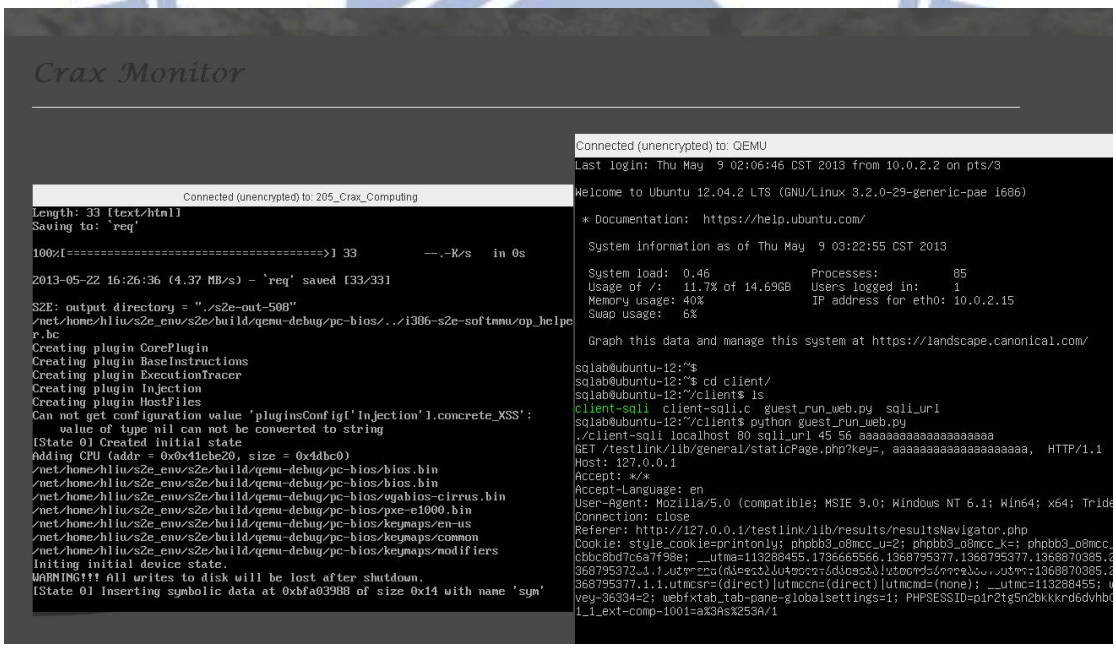


圖 20 CRAX 網頁監控介面

4-2 脅迫產生結果

於 4-2 中呈現使用 CRAX 分析 Ardilla 中四隻標竿程式的結果。表 4 中列出使用單一路徑擬真執行進行 SQL 資料隱碼攻擊檢測的結果，其輸入值為依據網頁表單而有所不同的字串，長度介於十五至三十之間。

表中 Number of crawled request 指網頁爬蟲所收集到的待測 URL 數量，而 Number of SQLi 是指該 URL 中偵測到 SQL 資料隱碼攻擊的弱點之數量。Number of all solved constraints 則是指在檢測過程中，所有經過的路徑限制式數量。

在表中可見即使路徑限制式的大小差異很大，然而對單一 URL 均可在一分鐘之內產生脅迫。待測網頁應用程式的 URL 越多，總共檢測需要的時間也越多。

Test Case	Schoolmate 1.54	Webchess 1.0.0rc2	Faqforge 1.3.2	EVE	Testlink 1.8.4	phpreci-piebook 2.24
Line of code	8125	6504	1710	904	144913	52631
CVE	-	-	-	-	2009-4238	2009-4883
# of crawled request	269	65	7	9	218	65
# of SQLi (vulnerable)	12	6	3	3	9	6
# of SQLi by MIT	6	12	1	2	-	-
Time per exploit	0.55 min	0.39 min	0.27 min	0.24 min	3.24min	4.89min
Time for all crawled requests	148.58 min	25.15 min	1.88min	2.12min	706.4min (30 TO)	315.2min (32 TO)
# of all solved constraints	952	15254	1104	934	18047	6322

# of crawled request: request may contain sensitive data / TO: Timeout

表 4 脅迫產生結果

4-3 與其他功能類似系統之比較

表 5 中呈現了 CRAX Web 和其他功能類似系統之比較。於表中可以看到，CRAX 使用單一路徑符號執行來產生脅迫，為一白箱測試系統，其跨站指令碼攻擊支援跨平台產生，而 SQL 資料隱碼攻擊目前僅實作於 PHP 平台之上。

Approach	Attacks/ Detectd	Generation Algorithm	W/B Box	Plateform
SAFELI[9], 2008	SQLI Attack	Statically inspect bytecode of application	WB	JAVA
Apollo[10], 2008	Malformed HTML Detect	Use Concolic execution to find bugs in PHP web applications	WB	PHP
Adrilla [11], 2009	XSS, SQLI Attack	It combines concrete and symbolic execution to covers paths	WB	PHP
Kudzu[12], 2010	XSS, SQLI Attack	Attack gramma and symbolic execution	WB	JavaScript
PIUIVT[14], 2010	XSS, SQLI Attack	Perturbation based Algorithm	WB	Java
[13], 2012	XSS, SQLI Detect	Dynamic taint analysis and symbolic execution	WB	Java
MySQLInjector [15], 2011	SQLIJ Attack	Blind SQL Injection based on True/False, Order by	BB	PHP
NKSI Scan [16], 2012	SQLIJ Attack	Modulize SQL Injection patten to generate attack string	BB	JSP, ASP
CRAX Web	XSS, SQLI Attack	Single path symbolic execution	WB	XSS: All, SQLI: PHP

* W/B Box: White/Black Box

表 5 自動產生/偵測網頁應用程式攻擊系統之比較 [19]

4-4 限制

由於 CRAX 是基於單一路徑擬真執行的自動脅迫產生器，因此無法偵測網站安全組態的不當設定 (OWASP 2013 TOP 10 A5) 此類偏向管理層面的網頁安全漏洞。同時，在設計概念中未考慮使用者的身分驗證問題，所以無法偵測使用者遭身分冒用的安全威脅 (OWASP 2013 TOP 10 A2)。

雖然使用單一路徑擬真執行能夠減少產生脅迫的時間，然而同樣也會對脅迫產生造成一定的限制。由於單一路徑擬真執行會紀錄所有測試資料經過的路徑，並利用來解出脅迫；亦即，執行脅迫所經過的路徑必定要與測試資料所經過的路徑相同。

換言之，若是脅迫執行的路徑和測試資料的路徑不同，即便此程式存在漏洞，CRAX 也無法偵測出來。

以圖 21 為例，受測網頁輸入 id＝AAAAA 之後會產生路徑限制式，在此稱為路徑限制式 a。CRAX 可以經由路徑限制式得到輸出字串 BBBBB，並且經由此路徑限制式產生其他輸入字串 CCCCC 。然而若此網頁漏洞與輸入資料之路徑限制式不同，如圖中脅迫執行之路徑限制式 b，CRAX 便無法經由路徑限制式 a 來偵測到此弱點。

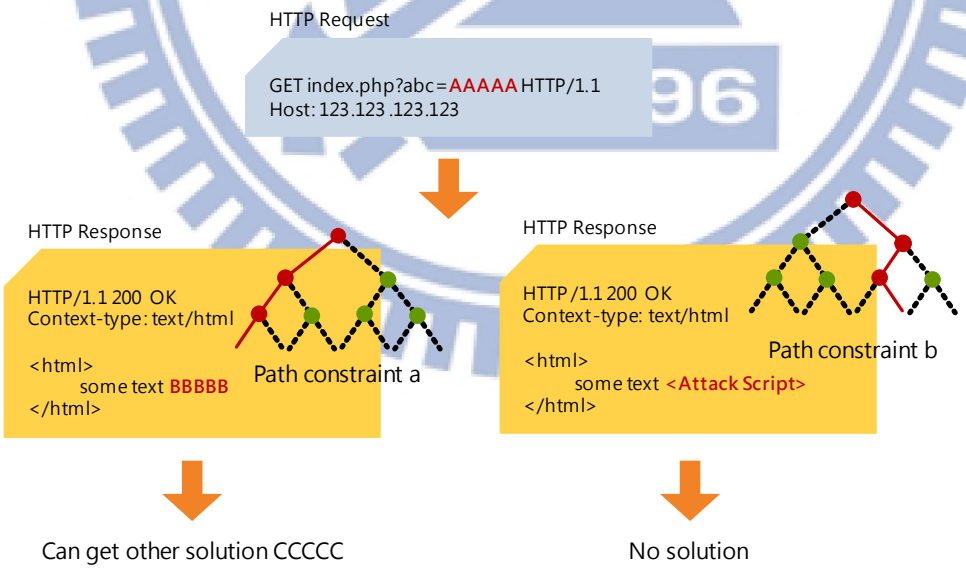


圖 21 脅迫產生限制示意圖

第五章　結論與未來展望

5-1　結論

本篇論文提出自動測試網頁程式弱點的方法。此方法基於單一路徑符號執行，需要該網頁應用程式的原始碼以建置符號環境下的測試環境，為白箱測試，同時也是一種動態測試。

此方法需要修改伺服器與資料庫間的連接器，目前僅針對 PHP 實作，理論上此方法具有跨平台特性，能夠擴展至所有語言。且此方法能應用於偵測常見的攻擊行為，例如 SQL 資料隱碼攻擊、遠端檔案包含攻擊(Remote File Inclusion)、指令注入攻擊(Command Injection) 等等。

5-2　未來發展方向

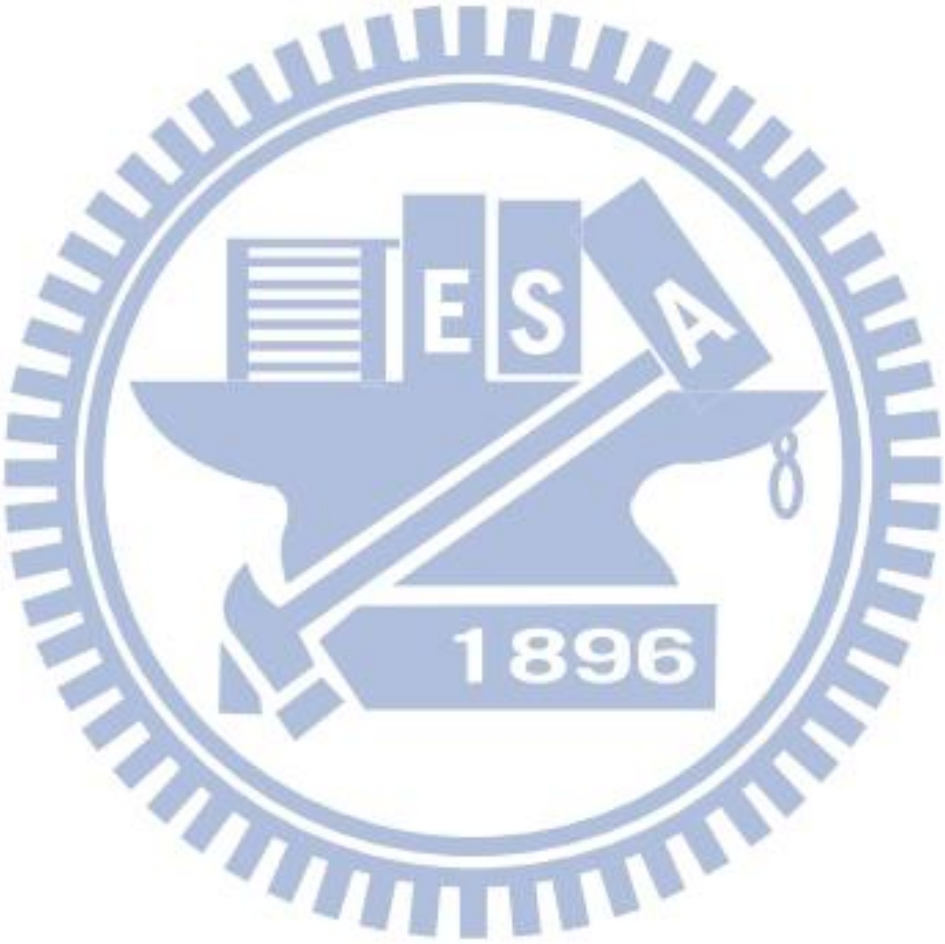
在本篇論文提出使用 CRAX 產生跨站指令碼與 SQL 資料隱碼攻擊的方式，理論上此方式能夠應用於遠端檔案包含攻擊與指令注入攻擊等。此類攻擊手法可透過修改對應的函式來偵測並產生對應的脅迫。

Other Web Security issues	Target Functions
Remote file Inclusion / Local File Inclusion	include(), include_once(), require(), requireonce()…
Directory traversal	fopen(), file(), unlink…
Command injection	system(), file()…
Code Injection	eval()…
File upload	move_uploaded_file(), rename(), …

表 6 常見網頁應用程式攻擊與相關函式列表

同樣的，在此僅實作 PHP 平台上的 SQL 資料隱碼攻擊，但是 S^2E 是一個支援所有語言及平台的符號執行引擎，基於 S^2E 實作的 CRAX 理論上只要在適當的函式中佈署符號資料偵測器，便能夠針對所有語言產生脅迫。未來 CRAX 能夠利用相同的方式對 JSP、Python、Ruby、ASP 等常

見的網頁應用程式用語言進行脅迫產生。



參考文獻

[1] King, J.C., *Symbolic execution and program testing.* Communications of the ACM, 1976. **19**(7): p. 385-394.

[2] Schwartz, E.J., T. Avgerinos, and D. Brumley. *All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)*. in *Security and Privacy (SP), 2010 IEEE Symposium on*. 2010. IEEE.

[3] Huang, S.-K., M.-H. Huang, P.-Y. Huang, C.-W. Lai, H.-L. Lu, and W.-M. Leong. *CRAX: Software Crash Analysis for Automatic Exploit Generation by Modeling Attacks as Symbolic Continuations*. in *Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on*. 2012. IEEE.

[4] Sen, K. *Concolic testing*. in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. 2007. ACM.

[5] Chipounov, V., V. Kuznetsov, and G. Candea, *S2E: A platform for in-vivo multi-path analysis of software systems.* ACM SIGARCH Computer Architecture News, 2011. **39**(1): p. 265-278.

[6] Bellard, F. *QEMU, a fast and portable dynamic translator*. 2005. USENIX.

[7] Cadar, C., D. Dunbar, and D. Engler. *KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs*. in *Proceedings of the 8th USENIX conference on Operating systems design and implementation*. 2008.

[8] OWASP. *The ten most critical web application security risks.* . Available from: https://www.owasp.org/index.php/Top_10_2013-Top_10.

[9] Fu, X. and K. Qian. *SAFELI: SQL injection scanner using symbolic execution*. in *Proceedings of the 2008 workshop on Testing, analysis, and verification of web services and applications*. 2008. ACM.

[10] Artzi, S., A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M.D. Ernst. *Finding bugs in dynamic web applications*. in *Proceedings of the 2008 international symposium on Software testing and analysis*. 2008. ACM.

[11] Kieyzun, A., P.J. Guo, K. Jayaraman, and M.D. Ernst. *Automatic creation of SQL injection and cross-site scripting attacks*. in *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. 2009. IEEE.

[12] Saxena, P., D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song. *A symbolic execution framework for javascript*. in *Security and Privacy (SP), 2010 IEEE Symposium on*. 2010. IEEE.

[13] Huang, Y.-Y., K. Chen, and S.-L. Chiang. *Finding Security Vulnerabilities in Java Web Applications with Test Generation and Dynamic Taint Analysis*. in *Proceedings of the 2011 2nd International Congress on Computer Applications and Computational Science*. 2012. Springer.

[14] Li, N., T. Xie, M. Jin, and C. Liu, *Perturbation-based user-input-validation testing of web applications.* Journal of Systems and Software, 2010. **83**(11): p. 2263-2274.

[15] Bashah Mat Ali, A., A. Yaseen Ibrahim Shakhatreh, M. Syazwan Abdullah, and J. Alostad, *SQL-injection vulnerability scanning tool for automatic creation of SQL-injection attacks.* Procedia Computer Science, 2011. **3**: p. 453-458.

[16] Tian, W., J.-F. Yang, J. Xu, and G.-N. Si. *Attack model based penetration test for SQL injection vulnerability*. in *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*. 2012. IEEE.

[17] Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *Hypertext transfer protocol–HTTP/1.1*, 1999, RFC 2616, June.

[18] 梁偉明, *自動化網頁測試與攻擊產生*, in *網路工程研究所* 2012, 國立交通大學:碩士論文 新竹市. p. 35.

[19] Alssir, F.T. and M. Ahmed. *Web Security Testing Approaches: Comparison Framework*. in *Proceedings of the 2011 2nd International Congress on Computer Applications and Computational Science*. 2012. Springer.