威脅模型風險評估機制之建構於安全軟體開發生命週期探討

A Study on Construction of Risk-Assessment Mechanism in Secure Software Development Life Cycle

陳振楠

中國科技大學大學資管學系教授

jn.chen3@gmail.com

伍台國

國防大學資訊管理學系教授

w13464@yahoo.com.tw

林宜隆

元培科技大學資訊管理系

中央警察大學資訊管理學系教授

cyberpaul747@mail.ypu.edu.tw

paul@mail.cpu.edu.tw

廖繹敦

國防大學管理學院資訊管理系研究生。

s942417@gmail.com

摘要

由於網際網路盛行，導致資安事件層出不窮，且攻擊手法已由 80 年代網路上的攻擊、90 年代電腦病毒，演變至今日 Web 應用軟體系統的安全漏洞攻擊。顯示傳統軟體開發生命週期(Software Development Life-Cycle，SDLC)已不足應付，應加入安全(Security)的觀念及檢核機制，並延伸至軟體安全開發生命週期(Secure Software Development Life-Cycle，SSDLC)，將「安全」擴大植入於軟體安全開發生命週期(SSDLC)各階段中，尤其在設計階段，可利用建構威脅模型(Threat Modeling) 的方法，來協助設計架構。威脅模型是一種以「安全」為主的分析方式，用來協助使用者分析網頁(Web)應用軟體系統所面對的最高安全風險來源，以及攻擊者如何執行他們攻擊的手法，使用威脅模型能瞭解 Web 應用軟體系統的弱點在何處。本研究以提出建構威脅模型風險評估機制，藉由實作階段的原始碼檢測反推建立設計階段的風險評估機制查核表，有助於開發人員在 Web 應用軟體系統成型前提昇軟體系統的安全性，並大幅降低 Web 應用軟體系統所付出的開發成本與遭受入侵的風險。

關鍵字：軟體開發生命週期、軟體安全開發生命週期、威脅模型、風險評估機制、原始碼檢測

Abstract

Web Application security is often considered as an add-on feature that can be incorporated until the software development life cycle is completed, but security protection is not equal to web Application security . It’s the vulnerabilities in Web Application make risk exposure, and let hackers and criminals have the opportunity to make software crash. We used

the concept of threat modeling to construct a risk assessment mechanism and build into the design phase of secure development life cycle. We found the mechanism can find vulnerabilities in software before it was built. By analyzing the information in design phases risk assessment mechanism can help software developer to consider security and adjust the basic framework in software development life cycle. We prove the mechanism can reduce the risk and improve the security of web application.

Keywords: Web Application security, software development life cycle, risk assessment mechanism

壹、緒論

近幾年來，資安威脅日趨複雜與多元，個資外洩事故頻傳，讓資安的重要性與日俱增，並廣受關注。從開放網頁應用程式安全計畫組織(Open Web Application Security Project，OWASP)所公佈的 2010 年的十大 Web 應用程式安全風險清單(OWASP Top 10)以及 CWE/SANS 所發佈的最危險的軟體錯誤清單(CWE/SANS Top 25)中可以發現許多的攻擊途徑,都是利用程式碼撰寫不當或是環境系統配置失當等弱點(Vulnerability)所形成的漏洞進行破壞，導致 Web 應用程式暴露於嚴重風險中，面臨各種可能的威脅與攻擊。

如果在安全軟體開發生命週期設計階段中建構威脅模型(Threat Modeling),進行 Web 應用程式弱點的風險評估，定義軟體分類的標準及安全性目標，並依據弱點影響的嚴重程度排定各個風險的順序，制訂相對應的風險管理計畫，採取風險紓緩行動(Mitigation Actions)，則能更有效地降低風險產生的機率以及減輕當風險轉變為問題帶來的衝擊。

本研究提出在安全軟體開發生命週期設計階段中,藉由建構威脅模型風險評估機制，期望在設計階段時能夠識別風險（Identify Risk）以及分析風險（Analysis Risk），從 Web 應用程式原始碼檢測（Source Code Analysis）分析報告中萃取出 Web 應用程式的風險因子，建構威脅模型為主軸的的風險評估機制；另外本研究的風險評估機制查核表可以做為開發人員的安全 Web 應用程式開發參考，有助於開發人員聚焦於 Web 應用程式的安全性。

貳、文獻探討

一、Web 應用程式安全相關議題

(一)Web 應用程式的安全目標

依據美國國家標準技術研究院(National Institute of Standards and Technology，NIST)所提出的文件「Risk Management Guide for Information Technology Systems」中，提到資訊系統的開發者及資訊的擁有者有責任去確保及維護所擁有資訊的機密性(Confidentiality，C)、完整性(Integrity，I)、可用性(Availability，A)。現今的 Web 應用程式為了提供更強大及更豐富的內容，使得 Web 應用程式的複雜度及可變性都較以往大幅增加；Web 應用程式威脅增多的原因可以歸納為 Web 應用程式特性改變、Web 應用程式環境安全性不足、駭客威脅增加等三種。

(二)傳統軟體開發生命週期設計階段可能產生的安全性漏洞

傳統軟體開發生命週期區分為需求分析階段(Requirement Phase)、設計階段(Design Phase)、實作階段(Implementation Phase)、測試階段(Test Phase)、發行與部署階段(Release and Deployment Phase)及維護階段(Maintenance Phase)六大階段，個階段執行過程僅著重在 Web 應用程式的功能性、整合性與執行效能，導致設計出的 Web 應用程式缺乏安全性，Web 應用程式的漏洞成為安全上的不定時炸彈。

表 1 所示為在設計階段因為規劃及設計不當而可能產生的安全漏洞。

表 1 Web 應用程式在設計階段可能產生的安全漏洞(本研究整理)

軟體開發生命週期階段名稱	可能產生的漏洞	
設計階段	外部資源設計	系統介面設計
	網路環境設計	功能介面設計
	運作環境設計	元件介面設計安全漏洞

軟體開發生命週期階段名稱	可能產生的漏洞
	內部資源設計安全漏洞　　邏輯控制安全漏洞 使用者介面設計安全漏洞

二、安全軟體開發生命週期方法論

安全軟體開發生命週期是將安全因素植入傳統軟體開發的各個階段中，在各個階段中導入安全性的思維，進行安全性的分析、規劃設計、與執行安全控制措施，有效達成軟體安全的目標。本篇研究蒐集安全軟體開發生命週期方法論的相關文獻，將其彙整並進行比對，如表 2 所示。

表 2 安全軟體開發生命週期方法論比較表(本研究整理)

提出者	名稱	步驟區分	特色
Gary McGraw	Touchpoint	需求與使用案例(Requirements and Use Cases) 設計與架構(Design) 測試計畫(Test Plans) 編寫程式碼(Code) 測試與測試結果(Test Result) 回饋(Field Feedback)	著重在風險管理、知識、軟體安全最佳實務。 將最佳實務彙整後可以區分為 7 類(Touchpoint)：程式碼檢測、風險分析、滲透測試、針對風險分析的安全測試、濫用案例、安全需求制定、安全操作。
Microsoft	Security Development Lifecycle(SDL)	訓練(Training) 需求(Requirements) 設計(Design) 實作(Implementation) 驗證(Verification) 發行(Release) 回應(Response)	以威脅模型為發展核心，共區分 14 個步驟。 其安全性原則為 SD^3+C：安全的設計 (Secure by Design)、安全的預設設定(Secure by Default)、安全的部署 (Secure in Deployment) 、溝通(Communications)
OWASP	Security Assurance Maturity Model(SAMM)	治理(Governance) 建構(Construction) 驗證(Verification) 部署(Deployment)	區分 12 項安全實踐步驟，每一個安全實踐都列出 3 個可以改善的目標。 為軟體開發商及廠商、線上服務提供商、金融服務機構以及政府單位定義範本。每一個產業範本代表不同的風險情形以及解決方法。

表 3 所示為 Touchpoint、SDL 以及 SAMM 三個方法論中各自設計階段的活動內容比較表，可以發現設計階段的安全性活動是以風險管理的角度為出發點。而本研究風險評估機制的建構由威脅模型及風險管理的概念作為出發點，除了探討設計階段中 Web 應用程式可能產生的安全性漏洞。

表 3 安全軟體開發生命週期設計階段活動內容比較表(本研究整理)

SSDLC 方法論名稱	設計階段活動內容
Touchpoint	1.風險分析(Risk Analysis) 記錄所有已知的風險，包含相關的觸發動作或是攻擊行為，並讓開發專案的所有參與者瞭解所發生的事情，接續依元件、階層、環境層級的順序進行風險分析，並量化風險。 2.外部審查(External Review) 藉由團隊之外的風險專家來協助進行風險分析。 3.以風險為基礎的安全測試(Risk-Based Security Tests) 根據風險分析的結果制訂安全測試計畫，安全測試需要從攻擊者和防禦者的角

SSDLC 方法論名稱	設計階段活動內容
	度進行，包含單元層級測試、部分系統測試以及整合後系統測試。
Security Development Lifecycle	1.建立設計需求(Design Requirements) 將 Web 應用程式的安全性及隱私性列為重要考量項目，定義 Web 應用程式的安全性目標。 2.減少攻擊面的產生(Attack Surface Reduction) 降低駭客利用 Web 應用程式漏洞發動攻擊的各種可能的機會，像是設定權限或是服務存取的限制，採用縱深防禦的方法阻擋可能的攻擊來源。 3.威脅模型(Threat Modeling) 藉由建構威脅模型的方式，分解 Web 應用程式，判斷可能的威脅，並藉由 STRIDE 方法將威脅分類，運用 DREAD 方法計算風險承擔值並排定優先處理順序。
Security Assurance Maturity Model	1.威脅評估(Threat Assessment) 依據正在開發 Web 應用程式的各項功能以及執行環境的各種情形，辨識及瞭解專案層面的風險。藉由分析每個專案可能面臨威脅的詳細資料，組織將更有效率地決定實施安全活動的優先順序。 2.安全需求(Security Requirements) 事先明訂 Web 應用程式的安全規範，提供內部及委外軟體開發團隊遵循。 3.安全架構(Security Architecture) 將安全性因素植入 Web 應用程式設計流程中，加強軟體設計流程，並利用設計框架、安全服務、設計範本等，讓組織擁有安全性架構可供參考使用，協助開發具有安全性的 Web 應用程式。

三、原始碼檢測與滲透測試

(一)原始碼檢測(Source Code Analysis)

原始碼檢測在執行的過程中，藉由剖析程式的原始碼，辨識程式中不同類型的敘述，並且偵測敘述的格式是否正確以及推理程式的控制流程，在許多情況下還會計算程式資料的所有可能值，可以與程式語言編譯器提供的錯誤偵測能力互補，檢測流程如圖 1 所示。

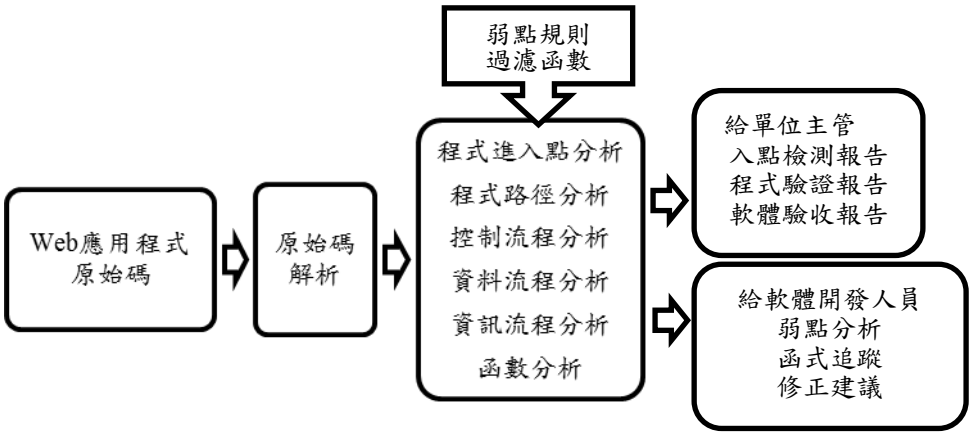


圖 1 原始碼檢測流程

(二)滲透測試(Penetration Testing)

滲透測試是一種在 SSDLC 的測試階段中所進行的安全性檢測活動，依據美國國家標準技術研究院(National Institute of Standards and Technology，NIST)所提出的文件「Guideline on Network Security Testing」，滲透測試的定義為：「滲透測試是測試者藉由對 Web 應用程式設計以及執行的瞭解，試圖繞過系統的安全防護以發動攻擊的一種安全測試，目的是藉由常用的工具及技術取得系統權限」，此種檢測方法為黑箱測試(Black-Box Testing)的一種。

(三)原始碼檢測與滲透測試效益比較

2009 年時，學者 Nuno Antunes 以及 Marco Vieira 針對原始碼檢測與滲透測試進行實驗分析，內容為利用三種原始碼檢測工具以及四種滲透測試工具檢測同一個 Web 應用程式的 SQL Injection 弱點，藉由檢測報告分析二者的弱點涵蓋率(Coverage)以及誤報率(False Positives)的差異，檢測結果發現使用原始碼檢測工具的弱點涵蓋率較使用滲透測試工具的弱點涵蓋率來的高，顯示原始碼檢測能比滲透測試找出更多的弱點。

四、Web 應用程式風險管理

好的風險管理是強調從「危機管理」到「預防管理」(Crisis Management to Anticipatory)。Web 應用程式風險管理做得好，可以避免損失、重做與矯枉過正，為了有效降低 Web 應用程式可能的風險以及產生的衝擊，開發人員必須學習去辨識、分析以及控制風險。依據學者 Linda Westfall 的研究中指出，風險管理的流程區分為風險識別(Identify)、風險分析(Analyze)、擬定風險管理計畫(Plan)、風險追蹤(Track)等步驟，流程如圖 2 所示。

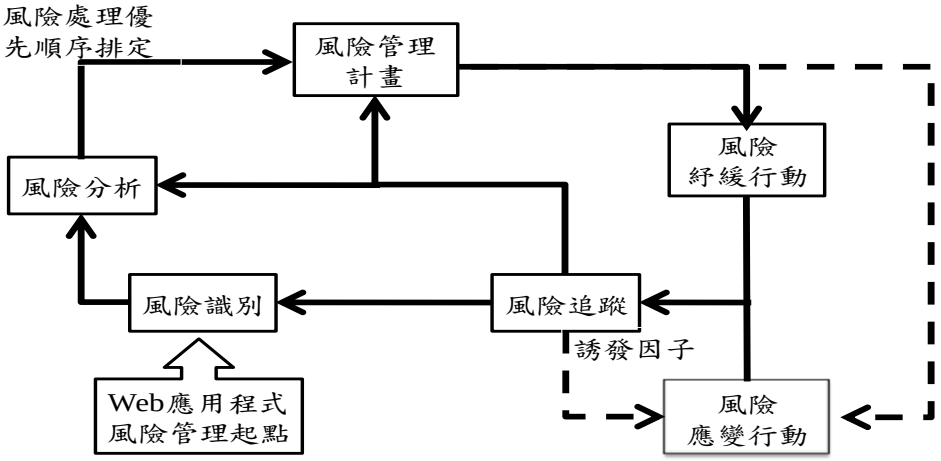


圖 2 風險管理流程(本研究整理)

參、建立 Web 應用程式風險評估機制

一、風險識別(Identify)

在風險識別的階段，主要研究對象限定為 2010 年 OWASP Top 10 的弱點項目，首先說明何謂開放網頁應用程式安全計畫組織(Open Web Application Security Project，OWASP)以及採用 OWASP Top 10 為風險識別對象的原因，並分析 Web 應用程式弱點形成的根本原因。

(一)開放網頁應用程式安全計畫組織

開放網頁應用程式安全計畫是一個開放社群、非營利性組織，其主要目標是研議協助解決 Web 應用程式安全之標準、工具與技術文件。OWASP Top 10 所公佈的弱點皆為現今相當普遍流行以及衝擊程度很高的弱點，不論是政府部門、企業組織以及個人都有可能遇到，因此本研究以 OWASP Top 10 作為風險識別對象。

(二)OWASP Top 10 弱點形成的根本原因

表 4 所示為 2010 年 OWASP 公佈的十大 Web 應用程式安全風險清單(OWASP Top 10)，歸咎這些弱點的肇因可以區分為「原始碼撰寫不當」以及「Web 應用程式的環境配置失當」等二類。「原始碼撰寫不當」是指 Web 應用程式開發者在程式碼撰寫時，忽略函式的處理或是未防範來自使用者可能的惡意輸入，導

致漏洞的產生，這一類的問題可以透過原始碼檢測協助發現可能的弱點；因為「Web 應用程式的環境配置失當」所造成的弱點指的是因為 Web 應用程式運行的周遭條件未妥善設定相關安全性參數或是採取安全措施，導致 Web 應用程式產生漏洞，例如在部署上線階段時，環境設定與參數配置有錯誤，造成 Web 應用程式暴露安全性風險，這一類的問題可藉由弱點掃描或是滲透測試發掘可能的弱點。

表 4 2010 年 OWASP 十大 Web 應用程式安全風險清單

弱點名稱	弱點形成原因
注入攻擊 (Injection)	原始碼撰寫不當
跨網站入侵字串攻擊 (Cross Site Scripting，XSS)	原始碼撰寫不當
遭破壞的認證與連線管理(Broken Authentication and Session Management)	Web 應用程式的環境配置失當
不安全的物件參考 (Insecure Direct Object Reference)	原始碼撰寫不當
跨網站冒名請求 (Cross Site Request Forgery，CSRF)	原始碼撰寫不當
網站安全組態不當設定(Security Misconfiguration)	Web 應用程式的環境配置失當
不安全的加密儲存 (Insecure Cryptographic Storage)	Web 應用程式的環境配置失當
未適當限制的 URL 存取(Failure to Restrict URL Access)	Web 應用程式的環境配置失當
不安全的傳輸防護(Insufficient Transport Layer Protection)	Web 應用程式的環境配置失當
未驗證的網頁重新導向(Unvalidated Redirects and Forwards)	Web 應用程式的環境配置失當

二、風險分析(Analyze)

(一)OWASP Web 應用程式風險評估項目

在 2010 年版本的 OWASP Top 10 中，風險清單的排名是依據弱點發生可能暴露的風險嚴重程度等級來排定，也就是考量風險發生的機率與影響的程度，風險的嚴重程度是由弱點所造成的衝擊乘上威脅發生的機率。用來計算量化風險嚴重程度的評估項目如表 5 所示，分別說明如下。

表 5 OWASP Top 10 Web 應用程式風險量化評估項目(本研究整理)

評估項目	說明/影響程度		
威脅曝光度	可分為組織內部人員、外部人員以及管理人員		
攻擊難易度(評量值)	簡單(1)	普通(2)	困難(3)
弱點的普遍程度(評量值)	廣泛流行(1)	常見(2)	不常見(3)
弱點偵測難易度(評量值)	容易(1)	一般(2)	困難(3)
技術衝擊程度(評量值)	嚴重(1)	中等(2)	輕微(3)
企業營運衝擊程度	對組織的利益、商譽、形象所造成的影響程度		

1.威脅曝光度(Threat Agent，TA)

威脅曝光度是指弱點可以被存取利用的難易程度，如果弱點僅能被組織內部利用，表示曝光度較低；若是弱點可以藉由網際網路被存取利用，表示曝光度較高，容易遭受外部人員利用。

2.攻擊難易度(Attack Vectors，AV)

攻擊難易度是衡量攻擊手法的難易程度以及多樣性，同時包含攻擊的工具是否容易取得，此評估項目程度可分為簡單、普通、困難。當攻擊手法愈簡單，表示該攻擊手法技術門檻較低或是工具較容易獲得。

3.弱點的普遍程度(Weakness Prevalence，WP)

此項目是評估弱點流行程度，區分為廣泛流行、常見及不常見三種。當弱點愈廣泛流行，表示 Web 應用程式遭受該風險的機率愈高。

4.弱點偵測難易度(Weakness Detectability，WD)

表示弱點的可偵測性，可以區分為容易、一般、困難。當弱點的偵測難易度為容易時，表示弱點較容易被偵測發現。

5.技術衝擊程度(Technical Impact，TI)

此項目評估攻擊者利用 Web 應用程式弱點成功發動攻擊時，可能對資源、資產、保管的資料與系統功能所造成的影響，依衝擊程度可區分為嚴重、中等、輕微三種。當技術衝擊程度為嚴重時，表示該弱點遭到利用時，其影響程度愈嚴重。

6.企業營運衝擊程度(Business Impact，BI)

此項目評估攻擊者利用 Web 應用程式弱點成功發動攻擊時，可能對組織的利益、商譽、形象所造成的衝擊程度。

風險承擔值(Risk Exposure，RE)的計算是以「攻擊難易度」、「弱點的普遍程度」、「弱點偵測難易度」三個數值加總後平均，乘上「技術衝擊程度」計算總分，以表示該項弱點之風險嚴重程度，如公式(1)所示，而「威脅曝光度」與「企業營運衝擊程度」此二項指標取決於 Web 應用程式的用途與企業政策。當計算出的風險承擔值愈低時，表示該弱點所造成的風險愈高。

公式(1)：
$$RE = \frac{AV + WP + WD}{3} \times TI$$

以弱點 Cross Site Scripting(XSS)為例做說明，依據 2010 年 OWASP Top10 的資料顯示，XSS 的「攻擊的難易度」為普通，評量值為 2，「弱點的普遍程度」為非常廣泛流行，評量值為 0，「弱點偵測難易度」為容易，評量值為 1，技術衝擊程度為中等，評量值為 2，將這些數值帶入公式(1)計算，可得風險承擔值為 2。

(二)計算風險承擔值

將 2010 年 OWASP Top 10 的風險評量值帶入公式(1)，可得到各個弱點的風險承擔值，如表 6 所示。

表 6 OWASP Top 10 弱點風險承擔值計算總表

弱點名稱	攻擊的難易度(評量值)	弱點的普遍程度(評量值)	弱點偵測難易度(評量值)	技術衝擊程度(評量值)	風險承擔值	弱點風險影響程度排名
注入攻擊(Injection)	簡單(1)	常見(2)	一般(2)	嚴重(1)	1 2/3	1
跨網站入侵字串攻擊 (Cross Site Scripting，XSS)	普通(2)	非常廣泛流行(0)	容易(1)	中等(2)	2	2
遭破壞的認證與連線管理(Broken Authentication and Session Management)	普通(2)	常見(2)	一般(2)	嚴重(1)	2	3
不安全的物件參考(Insecure Direct Object Reference)	簡單(1)	常見(2)	容易(1)	中等(2)	2 2/3	4
跨網站冒名請求(Cross Site Request Forgery，CSRF)	普通(2)	廣泛流行(1)	容易(1)	中等(2)	2 2/3	5
網站安全組態不當	簡單(1)	常見(2)	容易(1)	中等(2)	2 2/3	6

弱點名稱	攻擊的難易度(評量值)	弱點的普遍程度(評量值)	弱點偵測難易度(評量值)	技術衝擊程度(評量值)	風險承擔值	弱點風險影響程度排名
設定(Security Misconfiguration)						
不安全的加密儲存(Insecure Cryptographic Storage)	困難(3)	不常見(3)	困難(3)	嚴重(1)	3	7
未適當限制的URL存取 (Failure to Restrict URL Access)	簡單(1)	不常見(3)	一般(2)	中等(2)	4	8
不安全的傳輸防護(Insufficient Transport Layer Protection)	困難(3)	常見(2)	容易(1)	中等(2)	4	9
未驗證的網頁重新導向(Unvalidated Redirects and Forwards)	普通(2)	不常見(3)	容易(1)	中等(2)	4	10

三、風險管理計畫(Plan)

風險管理計畫包含制定紓緩行動以及制定應變行動兩部分，紓緩行動主要是藉由收集源碼檢測的分析報告，針對因為原始碼撰寫不當所產生的弱點進行剖析，並將檢測報告整理並萃取出Web應用程式弱點的風險因子，繪製成風險因子魚骨圖，製作風險評估機制查核表(Checklist)導入至SSDLC的的設計階段，降低Web應用程式風險發生的機率；應變行動則是當風險轉變為問題，也就是弱點已經遭到攻擊者利用，使用者在弱點修補完成前的空窗期應採取緊急措施保護現有Web應用程式，避免傷害持續擴大。

(一)風險紓緩行動

本研究是透過風險評估機制查核表的實施做為主要風險紓緩行動，而風險評估機制查核表的建立，是藉由收集Fortify原始碼白箱檢測工具的報告，萃取出弱點形成的風險因子，歸納綜整為Web應用程式威脅查核項目；首先介紹Fortify原始碼檢測軟體(Source Code Analyzer，SCA)。

1.Fortify原始碼檢測工具介紹

Fortify Source Code Analyzer(SCA)是Fortify公司推出的Web應用程式程式碼檢測工具，SCA可分析程式原始碼是否存在有安全性弱點，找出Web應用程式可能會執行的所有路徑，並指出弱點所在位置，SCA同時提供安全弱點相關資訊，包含問題追蹤流程、Call Graph協助問題分析及確認，方便開發人員瞭解弱點的發生位置，減少Web應用程式的修補時間。

2.風險評估機制查核表建立

本研究所收集到的源碼檢測報告計有八筆，分別以A、B、C、D、E、F、G作為原始碼檔案名稱，透過彙整各個原始碼檢測報告，可歸納下列結果：

(1)所收集到的8個Web應用程式的檔案中，全部被檢測出含有弱點。

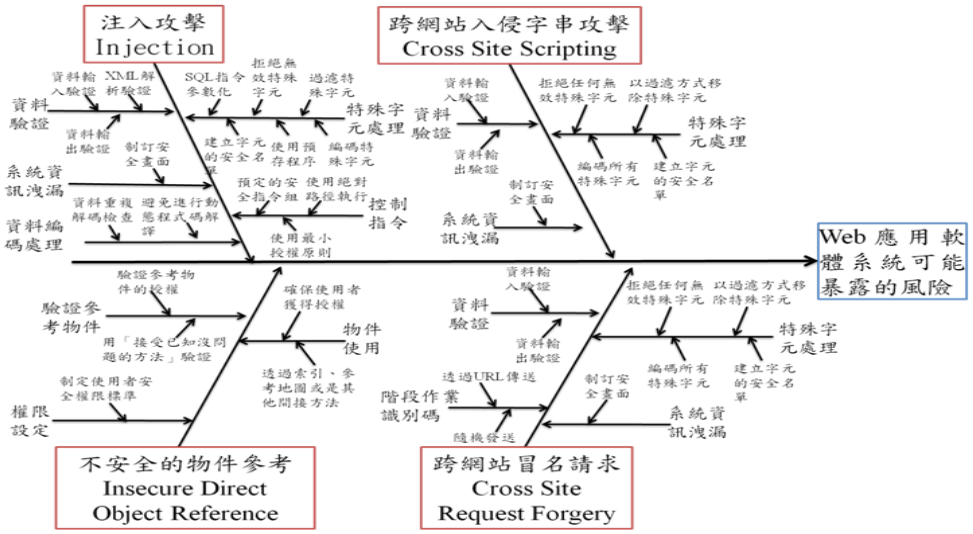
(2)本次共檢測89349行程式碼，弱點總數共計11571個，平均弱點密度為0.13，表示平均每撰寫100行程式碼，會產生13個弱點。

(3)弱點總數最多的Web應用程式為A網站，共檢測出9862個弱點，其次為D網站，檢測出597個弱點。

(4)由檢測結果可以發現 A1 注入攻擊弱點、A2 跨網站入侵字串攻擊弱點、A4 不安全的物件參考弱點、A5 跨網站冒名請求弱點等四項總計 6140 個，佔全部弱點總數比例的 53.06%，可得知因為「原始碼撰寫不當」所造成的弱點出現機率相當的高。

依據所收集的原始碼檢測結果報告，本研究分析並萃取出 Web 應用程式的風險因子，並繪製風險因子魚骨圖，如圖 3 所示。進一步分析風險因子可以發現資料驗證、特殊字元處理、系統資訊洩漏三個類別風險因子出現的頻率最高，表示若能在 SSDLC 的設計階段中能針對這三個風險因子提早預防及處理，將可以減少 Web 應用程式弱點的發生，降低風險暴露，大幅提昇 Web 應用程式的安全性。

圖 3 Web 應用程式風險因子魚骨圖



依據萃取出的 Web 應用程式弱點的風險因子，本研究綜整製作出 Web 應用程式設計階段風險評估機制查核表，如表 7 所示，查核項目共計三十六項，為一個以安全性為主要考量的查核表，協助開發人員於軟體開發生命週期的設計階段將安全性納入開發流程，瞭解弱點可能可能造成的風險嚴重程度，作為風險紓緩行動以及開發人員建構安全 Web 應用程式的依據。

表 7　Web 應用程式設計階段風險評估機制查核表

弱點類別	風險因子查核項目編號	風險因子查核項目	可能造成的風險嚴重程度	相對應的資訊安全目標	執行情形
注入攻擊	001	是否針對所有進入Web應用程式的資料進行輸入驗證，包含資料的長度、類型以及語法？	嚴重	機密性	是□否□
	002	是否針對所有從Web應用程式輸出的資料進行驗證？	嚴重	機密性	是□否□
	003	解析XML時，是否啟用驗證？	高	機密性	是□否□
	004	是否制訂安全畫面，避免因輸入錯誤而造成系統資訊洩漏？	嚴重	可用性	是□否□
	005	是否確認Web應用程式不會將相同的輸入資料解碼兩次？	嚴重	完整性	是□否□
	006	是否避免進行動態程式碼解譯？	高	完整性	是□否□
	007	是否將SQL指令參數化並建立連結參數？	嚴重	機密性	是□否□
	008	是否拒絕任何包含被視為無效特殊字元的輸入？	嚴重	機密性	是□否□
	009	是否以過濾方式移除特殊字	嚴重	機密性	是□否□

弱點類別	風險因子查核項目編號	風險因子查核項目	可能造成的風險嚴重程度	相對應的資訊安全目標	執行情形
		元？			
	010	是否建立字元的安全名單，允許可出現於資源名稱中的字元，並僅接受由被認可字元所組成的輸入？	嚴重	機密性	是□否□
	011	是否使用預存程序（Stored Procedures）？	嚴重	機密性	是□否□
	012	如果接受包含特殊字元的輸入，驗證動作是否編碼所有特殊字元？	嚴重	完整性	是□否□
	013	是否使用預定的安全指令組？	高	完整性	是□否□
	014	控制指令是否使用絕對路徑執行？	高	機密性	是□否□
	015	執行外部指令時，使否使用最小授權原則？	高	機密性	是□否□
跨網站入侵字串攻擊	016	是否針對所有進入Web應用程式的資料進行輸入驗證，包含資料的長度、類型以及語法？	嚴重	機密性	是□否□
	017	是否針對所有從Web應用程式輸出的資料進行驗證？	嚴重	機密性	是□否□
	018	是否制訂安全畫面，避免因輸入錯誤而造成系統資訊洩漏？	嚴重	可用性	是□否□
	019	是否拒絕任何包含被視為無效特殊字元的輸入？	嚴重	機密性	是□否□
	020	是否以過濾方式移除特殊字元？	嚴重	機密性	是□否□
	021	如果接受包含特殊字元的輸入，驗證動作是否編碼所有特殊字元？	嚴重	完整性	是□否□
	022	是否建立字元的安全名單，允許可出現於資源名稱中的字元，並僅接受由被認可字元所組成的輸入？	嚴重	機密性	是□否□
不安全的物件參考	023	是否驗證所用參考物件的授權？	嚴重	機密性	是□否□
	024	是否用「接受已知沒問題」的方法去驗證任何私有物件參考？	嚴重	機密性	是□否□
	025	是否制定使用者安全權限標準？	嚴重	機密性	是□否□
	026	是否避免使用者在未獲得授權之前取得或修改Web應用程式中的資料？	嚴重	可用性	是□否□
	027	是否透過索引、參考地圖或是其他間接方法避免將物件參考暴露給使用者？	嚴重	機密性	是□否□
跨網站冒名請求	028	是否針對所有進入Web應用程式的資料進行輸入驗證，包含資料的長度、類型以及語法？	嚴重	機密性	是□否□
	029	是否針對所有從Web應用程式輸出的資料進行驗證？	嚴重	機密性	是□否□
	030	是否透過URL傳送階段作業識別碼，而不是使用Cookie的Web	嚴重	機密性	是□否□

弱點類別	風險因子查核項目編號	風險因子查核項目	可能造成的風險嚴重程度	相對應的資訊安全目標	執行情形
		應用程式			
	031	階段作業的識別碼是否為隨機發送？	嚴重	完整性	是□否□
	032	是否拒絕任何包含被視為無效特殊字元的輸入？	嚴重	機密性	是□否□
	033	是否以過濾方式移除特殊字元？	嚴重	機密性	是□否□
	034	如果接受包含特殊字元的輸入，驗證動作是否編碼所有特殊字元？	嚴重	完整性	是□否□
	035	是否建立字元的安全名單，允許可出現於資源名稱中的字元，並僅接受由被認可字元所組成的輸入？	嚴重	機密性	是□否□
	036	是否制訂安全畫面，避免因輸入錯誤而造成系統資訊洩漏？	嚴重	可用性	是□否□

陸、結論

為了有效提昇 Web 應用程式的安全性，本研究以威脅模型為主軸，建構安全軟體開發生命週期風險評估機制，協助在設計階段中，在 Web 應用程式開發初期即有效管制與監控安全弱點，再配合原始碼檢測以及滲透測試等安全性檢測工具，找出安全性的漏洞進行弱點修補，確保 Web 應用程式具備高度安全性，有效減少安全性弱點，以達到 Web 應用程式機密性、完整性以及可用性的安全性目標。本研究提出的安全軟體開發生命週期評估機制具有六項優勢：

1. 降低 Web 應用程式風險發生的機率

本研究提出的風險評估機制查核表可以於軟體開發生命週期設計階段協助找出 Web 應用程式潛在的風險因子，較實作階段的白箱測試或是測試階段的黑箱測試提早發現 Web 應用程式的弱點以及漏洞。

2. 確保 Web 應用程式達到安全性目標

本研究從原始碼檢測報告中萃取出的風險因子製作風險評估機制查核表，列入 Web 應用程式安全設計原則，有助於開發人員在 Web 應用程式成型前提昇 Web 應用程式的安全性。

3. 降低 Web 應用程式的漏洞修補成本

本研究於軟體開發生命週期設計階段導入威脅模型的概念，在 Web 應用程式部署前找出並改正設計流程上存在的風險因子，降低 Web 應用程式上線後安全性漏洞修補所需耗費的時間及人力成本。

4. 強化開發人員安全認知教育

藉由本研究的風險評估機制，可以用來教育訓練開發人員 Web 應用程式的安全性素養，幫助開發人員思維的改變，強化開發人員的安全認知教育。

5. 幫助開發人員聚焦與經驗分享

本研究針對 Web 應用程式特性做專屬的分析，找出 Web 應用程式的風險因子，彙整相關風險因子訊息建構成「安全 Web 應用程式開發指引」，有助於開發人員聚焦於 Web 應用程式的安全性。

6. 將安全軟體開發生命周期各階段的安全性活動更緊密結合

本研究的查核表，不但大幅提昇安全性，更讓開發團隊能夠緊密思考安全性的活動，增強 Web 應用程式安全防護的強度。

本研究未來可持續蒐集 WEB 應用程式案例，以佐證威脅模型風險評估機制於安全軟體開發生命週期探討可用性。

參考文獻

行政院研究發展考核委員會，97 年度國家資通安全技術服務與防護管理委外服務案-Web 應用程式安全參考指引 V.2，2009 年。

林宜隆、賴森堂，提昇軟體安全性的安全品質測試模式，2005 網際網路：資安、犯罪與法律社會學術研究暨實務研討會，台北：國立台灣師範大學綜合大樓國際會議廳，2005 年 12 月 23 日。

蔡震天，網頁 Web 應用程式原始碼弱點分析之研究－以淡江大學為例，淡江大學資訊管理學系，碩士學位論文，2010 年。

潘天佑，資訊安全概論與實務，2008 年，台北：碁峰資訊股份有限公司。

賴森堂，安全軟體的建置關鍵—靜態檢視與動態測試技術，風險社會與安全管理學術研討會，台北：政大公企中心國際會議廳(中央警察大學與政治大學主辦)，2006 年 12 月 14 日。

賴森堂，以介面設計為基礎的軟體安全品質量測模式，中華民國品質學會第 43 屆年會暨第 13 屆全國品質管理研討會，台北：台大集思會議中心(中華民國品質學會、國立新竹教育大學、國立教育研究院籌備處主辦)，2007 年 11 月 10 日。

楊博閔，建構安全軟體開發生命週期風險評估之探討—以風險管理的流程提出以威脅模型建構風險評估機制，2011TBI 商管與資訊研討會(台北大學)，2010 年 4 月 28 日

Ian Sommerville，陳玄玲譯，軟體工程-軟體開發技術與軟體專案管理，2009 年，台北：台灣培生教育出版股份有限公司。

Michael Howard，David C.LeBlanc，林正平譯，防駭程式撰寫實務第二版，台北：文魁資訊股份有限公司，2006 年，

Committee on National Security Systems, National Information Assurance Glossary, 2010.

Gary McGraw, Software Assurance for Security”, IEEE Computer, Volume 32, Issue 4, Apr 1999, pp.103-105.

Gary McGraw, Managing Software Security Risks, IEEE Computer, Volume 35, Issue 4, Apr 2002, pp.99-101.

Gary McGraw, Software Security, IEEE Security & Privacy, Volume 2, Issue 2, Mar-Apr 2004, pp.80-83.

Gary McGraw, Misuse and Abuse Cases：Getting Past the Positive, IEEE Security & Privacy, Volume 2, Issue 3, May-June 2004, pp.90-92.

Gary Stoneburner, Alice Goguen, Alexis Feringa, Risk Management Guide for Information Technology Systems, National Institute of Standards and Technology, 2002.

IBM X-Force® research and development teams, IBM X-Force® 2010 Mid-Year Trend and Risk Report, 2010.

John Steven, Threat Modeling—Perhaps It’s Time, IEEE Security & Privacy, Volume 8, Issue 3, May-June 2010, pp.83-86.

John Wack, Miles Tracy, Murugiah Souppaya, Guideline on Network Security Testing, National Institute of Standards and Technology, 2003.

Karen Mercedes Goertzel, Theodore Winograd, Holly Lynne McKinley, Patrick Holley, Booz Allen Hamilton, Security in the Software Life Cycle, Version 1.2 (Draft)，Department of Homeland Security, 2006.

Linda Westfall, The Certified Software Quality Engineer Handbook, ASQ Quality Press，pp.282～303, 2009.

Microsoft Corporation, Simplified Implementation of the Microsoft SDL, 2010.

Nuno Antunes, Marco Vieira, Comparing the Effectiveness of Penetration Testing and Static Code Analysis on the Detection of SQL Injection Vulnerabilities in Web Services, 2009 15th IEEE Pacific Rim International Symposium on Dependable Computing, 2009.

Pravir Chandra, Software Assurance Maturity Model V1.0, 2009.

OWASP, OWASP Top10-2010 The Ten Most Critical Web Application Security Risks,2010.

Richard Ford , Michael Howard , Revealing Packed Malware, IEEE Security & Privacy , Volume 6 , Issue 5 , Sept-Oct 2008, pp.65-69.

Stone-Gross Brett, Cova Marco, Gilbert Bob, Kemmerer Richard, Kruegel Christopher, Vigna Giovanni, Analysis of a Botnet Takeover, IEEE Security & Privacy, Volume 9, Issue 1, Jan-Feb 2011, pp.64-72.

Steve Christey, 2010 CWE/SNAS Top 25 Most Dangerous Software Errors V1.06, Sept 2010.