前言

歡迎來到針對真實世界的威脅建模實務指南，本書是我們 10 年來在各自職業生涯中對威脅建模和安全系統設計進行研究、開發和實踐的成果。我們努力確保本書的內容不只透過我們的觀察，也有來自於我們同事的經驗和應用安全社群成員的支持。

儘管我們試圖在本書中展示所有當前技術及具前瞻性方法，但是因為威脅建模是一個仍在不斷發展的領域，未來幾個月、甚至幾年內的變化將超越本書所討論的內容。在撰寫本書的當下（2020），雖然已經存在可用於執行安全建模與分析的數十種不同方法，但全球各個專注於安全技術的活躍論壇和開發者社群，仍持續地發明新技術或更新現有的方法。考慮到這一點，本書的目標是基於足夠的理論和指導，為你提供可執行且易於查找的資訊，讓你得出自己的結論，並為你的團隊和系統調整這些技術。

撰寫動機

通常人們普遍會認為只有是資安領域內的專家，才有資格執行威脅建模，但實際上並非如此。威脅建模應該是指程式開發過程中的基本準則及確保安全性的機制。因此，我們在本書中的終極目標是改變人們對威脅建模的看法，使其成為任何人都可以易於上手、學習和執行的學科。

那為何是由我們開始呢？多年前，我們和你們許多人一樣：對整個「威脅建模」感到困惑。

隨著時間的推移，我們逐漸了解一些方法、許多痛點以及一個好的威脅模型可以帶來的許多樂趣。在這個過程中，我們遇到了許多有趣、聰明的人，他們將威脅建模（以及隨之而來的元知識）提升到了一個全新的水平，我們不僅向他們學習，也發展出自己的點子並意識到我們可以在此過程中幫助其他人，讓他們擺脫對威脅建模的恐懼、不確定性和對威脅建模的懷疑。簡而言之：我們希望人們和我們一樣對此感到興奮。

目標讀者

我們為那些負責提高其系統安全性設計、更安全的開發過程和系統安全性更新的開發團隊成員（開發人員、架構師、設計人員、測試人員、DevSecOps）編寫了這本書。這包括設計、構建或維護產品的工程師或 IT 系統人員。

傳統的安全從業者也會在本書中找到價值——尤其是那些還沒有威脅建模經驗的人——雖然本書的目標受眾主要是針對系統開發團隊人員，但產品經理、專案經理等等比較不那麼技術導向的職能部門應該也能在本書找到靈感，起碼可以在這個過程中了解自己的價值和定位！

本書包含（或不包含）的論點

我們主要關注如何使用威脅建模來分析系統設計，以便你可以識別系統實作中和部署時固有的風險，並在一開始就可以避免它們。我們不會提供針對安全設計或分析特定拓撲、系統或演算法的手把手教學步驟；相反地，我們引用了其他在這些方面做得很好的書籍給你參考。我們的目標是為你提供識別存在風險條件所需的工具，為你提供解決這些風險條件的具體方法選項，並為你提供更多資訊的來源，以幫助你擴展威脅建模技能。

在〈導論〉中，我們提供安全原則和安全設計技術的背景知識，並討論了保護你的數據和系統功能的基本屬性和機制。我們細細檢查安全保護、隱私和安全狀態之間的關係，並定義什麼稱作**風險**。我們也一併確認了在你的系統中決定風險的因素。對於那些剛接觸應用程式安全的人、以及那些希望重新了解原則和目標的人來說，〈導論〉中涵蓋的安全基礎知識尤其重要。

在第 1 章中，我們著重於系統建模技術，並向你展示如何識別對評估系統安全性至關重要的關鍵特徵。我們識別哪些是可被利用的弱點，以及這些弱點有哪些對系統安全產生負面影響的方法。

在第 2 章和第 3 章中，我們概述威脅建模並將它視作為系統開發生命週期中的其中一環，且深入回顧現在流行的威脅建模技術，以便在你進行建模和分析系統時可以派上用場。此外，我們還討論了那些處於發展中的新方法和將威脅建模過程遊戲化的主題。

第 2 章及其後續內容對所有讀者都很有價值，包括那些已經了解為什麼威脅建模是一項至關重要的活動、並掌握安全設計原則的經驗豐富安全從業人員。

在第 4 章和第 5 章中，我們討論了威脅建模方法、自動化和敏捷開發（包括 DevOps 自動化）的未來。我們還介紹了以新穎有趣的方式執行威脅建模的專業技術。這些章節對於進階讀者來說應該特別有趣。

第 6 章介紹了我們經常從開發團隊聽到的常見問題，而這些問題亦常出現於採用威脅建模的組織或團隊。有鑑於此，我們提供建議和指導，幫助使用者在威脅建模過程中取得進步，避免常見的陷阱和成功的障礙。

附錄 A 包含使用 pytm 構建和分析系統模型威脅的完整範例。附錄 B 包含威脅建模宣言——一個關於在今天的系統部署中，是什麼使得威脅建模變得有價值且必要的方向聲明。

這些技術適用於各種系統

在本書中，我們以基於軟體的系統為特色，因為它們在所有場景中都是通用的，而且我們不希望將物聯網（IoT）或雲端技術等知識作為理解示例的先決條件。但是我們討論的技術適用於所有系統類型，無論是基於硬體、雲端，還是負責將資料從這一端傳到另一端以安全儲存的系統和軟體，不管是怎麼樣的系統類型與技術組合，我們所討論的威脅建模方式都適用。我們甚至提供分析業務流程的指導，以幫助你了解它們對系統的影響。

你的貢獻很重要

如果你閱讀過其他有關如何進行威脅模型的文章，你可能會注意到我們提供的技術、建構過程和實作方法的意見略有不同，這是設計使然（沒有雙關語！）。如果我們所做的努力在威脅建模社群內引發關於如何理解安全性及其對系統設計影響的建設性辯論，威脅建模將會得到改善，而依賴它的個人也將從中受益。正如我們所說，威脅建模在不斷發展，我們鼓勵你為它的發展做出貢獻。也許有一天我們會在研討會上與你會面或與你一起開發專案，討論我們的經驗並相互學習。

你可以透過 *https://threatmodeling.dev* 提交請求來聯繫我們。

導論

你們了解知識的方式就是我想知道的內容。

——Richard Feynman，美國物理學家

在〈導論〉中，我們將解釋威脅建模的基礎知識。作於評估安全性的基礎知識，我們還會介紹你所需要了解的重要安全原則，以便你分析系統的安全性。

威脅建模基礎

讓我們從鳥瞰圖開始了解什麼是威脅建模和它為什麼是有用的工具，以及它如何融入開發生命週期和整體安全計畫。

什麼是威脅建模？

程式開發的過程中往往因為不太理想的設計選擇導致軟體弱點的產生，而**威脅建模**（*Threat Modeling*）是藉由分析系統來尋找這些弱點的過程。威脅建模的目標就是在這些弱點被納入系統（程式開發或部署的結果）之前識別它們，以便你可以儘早採取糾正措施。威脅建模活動是一項概念性實戰，旨在幫助你了解應該修改系統設計的哪些特徵，以便將系統中的風險降低到系統擁有人、系統使用者和系統管理員可以接受的程度。

在執行威脅建模時，你需要將系統視為一群組件的集合，及此集合與系統外部世界的交互行為（例如與之交互的其他外部系統）以及可能在這些系統上執行操作的使用者行為集合。然後你試著想像這些系統組件集合和它們之間的交互行為，可能會有哪些失敗的情境發生。在此過程中，你將識別對系統的威脅，隨著這個過程，你會對系統進行修改並再反覆檢驗威脅，直到最後，系統的最終結果是可以抵抗你所想像的各種威脅。

讓我們開門見山地說：威脅建模是一個持續循環的過程。它以明確的目標開始，然後是分析和採取相對應的處置行動，然後再重複這一個過程。威脅建模並不是萬靈藥——它並不能解決你的所有安全問題。威脅建模也不是一個指向網站或是程式碼儲存庫的雷達，一鍵生成可供勾選的項目清單列表。如果你讓團隊中適當的人員參加，則威脅建模是一個合乎邏輯推導且充滿智慧的過程。它會將設計考量和執行步驟進行充分討論，並使其更為清晰與完整。這些過程都需要團隊成員一同付出工作時間與專業知識來參與。

威脅建模的第一條規則可能是古老的格言「垃圾進，垃圾出」（garbage in, garbage out）[1]。如果你將威脅建模作為團隊工具箱的一部分，並讓每個人都以積極的方式參與其中，你將獲得很多好處；反之，如果你並沒有充分理解威脅建模的強處與弱項，或並沒有全心全意的投入其中，只把它當成一個「檢查項目表」，那這個過程就只是一個浪費時間的行為。一旦找到適合你和你團隊的方法，並付出必要的努力使其發揮作用，你的系統整體安全狀況將大幅增長。

為什麼需要威脅建模

從長遠的角度而言，威脅建模將使你的工作更輕鬆、更安全，它將帶來更清晰的架構，更明確定義的信任邊界（你還不知道它們是什麼以及它們為什麼重要，但很快你就會知道！）集中的安全測試和更好的文檔，這些都是你需要威脅建模的最佳原因。最重要的是，它將以有組織、精心策劃的方式向你和你的團隊灌輸安全意識的超能力，從而在你的開發工作中產生更好的安全標準和指導方針。

儘管這些附帶好處很重要，但它們並不是最重要的。了解你的系統中可能出現的問題以及你可以透過做些什麼來增加你對所交付內容的信任，這才能讓你可以自由地專注於系統的其他方面。這就是威脅建模需求背後的真正原因。

相同地，正確理解你不需要威脅建模的原因，也是很重要的事情。威脅建模並不能解決你所有的安全問題；它也不會立即將你的團隊轉變為安全專家。最重要的是，你不需要透過使用它來符合什麼安全規範。如果執行安全性實踐措施只是為了在符合規範的檢查表中打勾，那比什麼都沒做更令人沮喪。

1 這句話被 Wilf Hey 和陸軍專家 William D. Mellin 認可。

系統開發生命週期中的威脅建模

威脅建模是在系統開發生命週期中執行的一項活動，對系統的安全性至關重要。如果不以某種方式執行威脅建模，則你所選擇的設計，很有可能容易被利用以至於引入安全故障，並且以後肯定很難修復（且成本高昂[5]）。為了與「內建而非附加」的安全原則保持一致，威脅建模不應被視為合規性里程碑；在最關鍵的時刻未能執行此活動會帶來現實世界的後果。

如今，大多數成功的公司不再像幾年前那樣執行專案。例如，無伺服器計算[6]等開發範式，或 CI/CD[7] 中的一些最新趨勢和工具，對開發團隊設計、實作和部署當今系統的方式產生了深遠的影響。

由於市場需求和爭先恐後的競爭，如今你很少有機會能在系統開發之前坐下來查看設計是否足夠完整。產品團隊依靠「最小可行性產品」的版本向公眾介紹他們的新想法，並開始建立品牌和追隨者。然後，他們依賴追加發布的方式來添加功能至產品內，並在出現問題時進行更改，這種做法會導致在開發週期的後期對設計進行**重大**更改。

現代系統具有前所未有的複雜性。你可能會使用許多第三方元件、函式庫和框架（可能是開放原始碼或封閉原始碼）來構建你的新軟體，但這些元件很多時候都缺乏文檔記錄、難以理解和缺少安全保障。要創建「簡單」的系統，你需要依賴複雜的軟體、服務和功能的層層堆疊。同樣地，以無伺服器計算部署為例，說「我不在乎執行環境、函式庫、實體機器或網絡，我只關心我的功能是否正常」這樣的話是短視近利的。這一切的幕後隱藏著多少軟、硬體細節？你對你的系統功能「底下」發生的事情有多少控制權？這些事情如何影響系統的整體安全性？你如何驗證你使用的是最合適的角色和訪問規則？

要可靠地回答這些問題並立即獲得結果，你可能會想要聘請外部安全專家。但是安全方面的專業知識可能會依不同專家的能力而有所不同，而且聘請專家的費用可能很高。一些專家專注於特定的技術或領域，而另一些專家的關注範圍廣泛但不夠深入。當然，並不是每位顧問都是這種情況，我們將是第一個證明我們在威脅建模顧問方面擁有一些豐富經驗的人。但是，你可以看到，無論是否聘僱外部專家，但發展適合自身開發團隊的威脅建模知識，仍是一件相當有助益的事情。

---

5 Arvinder Saini，「在 SDLC 的每個階段修復錯誤的成本是多少？」，軟體完整性部落格，概要，2017 年 1 月，*https://oreil.ly/NVuSf*；Sanket，「修復錯誤的指數成本」，DeepSource，2019 年 1 月，*https://oreil.ly/ZrLvg*。

6 「什麼是無服務器計算？」，Cloudflare，2020 年 11 月訪問，*https://oreil.ly/7L4AJ*。

7 Isaac Sacolick，「什麼是 CI/CD？持續整合和持續交付的解釋」，InfoWorld，2020 年 1 月，*https://oreil.ly/tDc-X*。

開發安全的系統

無論使用何種開發方法，你的系統開發方式都必須經過一些非常具體的階段（見圖 I-1）。

- 開始想法
- 設計
- 實作
- 測試
- 部署

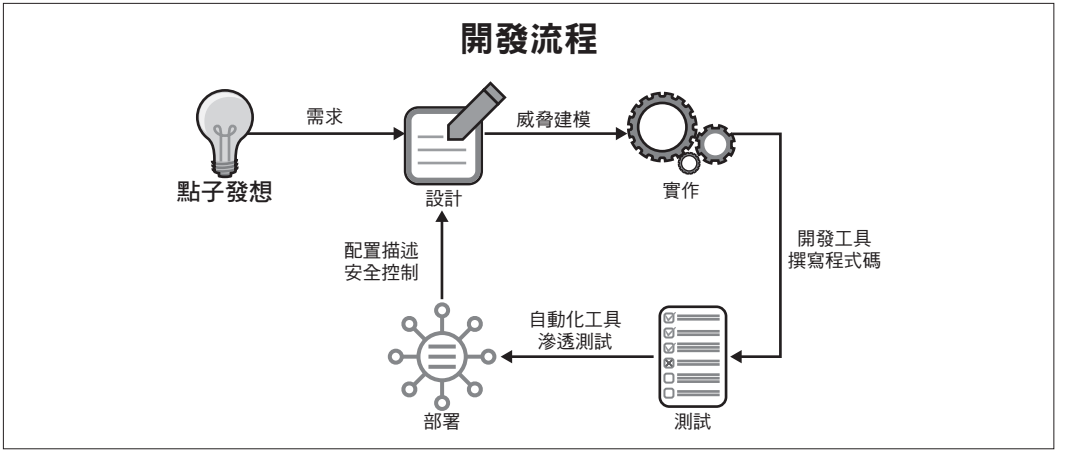

圖 I-1　開發循環和相關的安全活動

例如，在瀑布式開發方法中，這些階段自然而然地相繼發生。請注意，文件的作用是持續不斷的——它必須與其他階段同時發生才能真正有效。使用這種方法時，很容易看出威脅模型在設計階段時可以提供最大的好處。

你肯定會在本書中多次看到我們精心地將威脅建模與設計聯繫起來。這是為什麼？

一個經常被引用的概念表明[8]，越接近部署階段或完成部署之後，其解決問題的成本就會顯著地增加。這對於熟悉製作和行銷軟體的人來說是顯而易見的；相較於導入解決方案於已經部署在數千個或數百萬個地點的系統（某些極端情況下）[9]，將解決方案應用於開發

8　Barry Boehm，**軟體工程經濟學**（Prentice Hall，1981 年）。

9　凱拉 · 馬修斯（Kayla Matthews），「物聯網駭客會給經濟造成什麼損失？」，人人享有物聯網，2018 年 10 月，*https://oreil.ly/EyT6e*。

中的系統要來得便宜得多。你不必處理某些用戶未應用修補檔的責任，或因修補系統而引入的向後兼容性可能失敗的問題。你不必與因某種原因而無法繼續使用修補檔的用戶打交道，也不必承擔支持冗長且有時不穩定的升級過程的成本。

因此，威脅建模本質上是著眼於**設計**，並試圖識別**安全缺陷**。例如，如果你的分析表明某種訪問模式使用的密碼是寫死在程式內，則它會被識別為需要解決的**問題**。如果該發現未在設計階段得到解決，那在系統生命週期後期，你可能正在處理一個將被利用的問題。這也稱為具**被利用性**的**漏洞**並且有一定**機率**發生，如果被利用的話，則需要付出相關**代價**。你也可能無法識別問題，或者無法正確地確認可以被利用的內容。完美和完整不是本練習的目標。

威脅建模的主要目標是識別缺陷，使它們成為**發現**（你可以解決的問題）而不是漏洞（可以被利用的問題）。然後，你可以應用緩解措施來降低被利用的可能性和被利用的成本（即損害或影響）。

一旦你確定了一個發現，你就會採取行動來**減輕**或糾正它。你可以透過應用適當的控制來做到這一點；例如，你可以創建一個動態的、用戶定義的密碼，而不是寫死在程式內的密碼。或者，如果情況允許的話，你可以針對該密碼運行多個測試以確保其強度；或者你可以讓用戶決定其密碼設置策略；或者，你可以完全改變做法，藉著不使用密碼並支援 WebAuthn[10]，以完全移除缺陷。在某些情況下，你可能只是承擔風險——考量系統的部署方式，你決定使用寫死在程式內密碼可能沒有問題。（提示：它不是真的沒問題。真的，再想想看。）有時你必須確定風險是可以接受的。在這些情況下，你需要記錄發現、確定和描述不解決它的理由，並將其作為威脅模型的一部分。

重要的是要強調（我們將在整本書中回到這一點）威脅建模是一個進化過程。第一次分析時，你可能無法發現系統中的所有缺陷。例如，你可能沒有合適的資源或合適的利益相關者來檢查系統。但是擁有一個初始威脅模型比根本沒有威脅模型要好得多，等到下一次迭代，當更新威脅模型時，會更好地識別其他缺陷，並提供更高級別的保證——即沒有發現缺陷。你和你的團隊將藉此獲得經驗和信心，這將引導你考慮新的、更複雜或更微妙的攻擊與向量，而你的系統將會不斷地改進。

別再使用瀑布式開發

讓我們轉向更現代的敏捷開發和 CI/CD 方法。

10 「什麼是 WebAuthn？」 Yubico，*https://oreil.ly/xmmL9*。

基本安全原則

導論的其餘部分簡要概述了基礎安全概念和術語，這些概念和術語對於開發團隊和安全從業者至關重要，至少有一定程度的熟悉。如果你想更了解這些原則，請查看我們在本章和本書中提供的許多優秀參考資料。

作為個人或團隊，在學習安全知識的旅途中——熟悉這些原則和術語是關鍵基礎。

基本概念和術語

圖 I-2 強調了系統安全中的關鍵概念。理解這些關係和安全術語，是理解為什麼威脅建模對安全系統設計至關重要的關鍵。

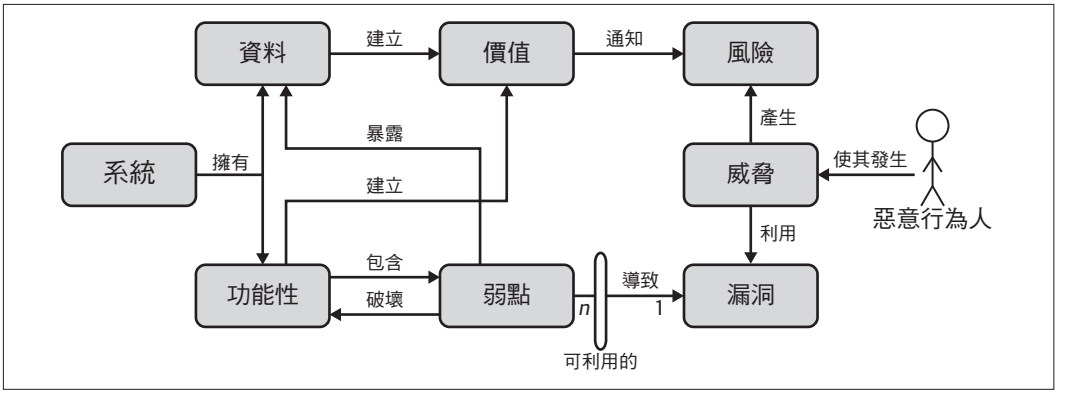

圖 I-2　安全術語的關係

*系統*包含資產——用戶依賴的*功能*，以及系統接受、儲存、操作或傳輸的*資料*。系統的功能可能包含缺陷，也稱為*弱點*。如果這些弱點是可被利用的，這意味著如果它們容易受到外部影響，則被稱為*漏洞*，利用它們可能會使系統的操作和資料面臨暴露的風險。*參與者*（系統外部的個人或程序）可能具有惡意，並且如果存在某種條件，使得這樣的行為成為可能發生的事，行為者可能會嘗試利用該漏洞；一些技術嫻熟的攻擊者能夠改變條件，創造嘗試利用的機會。在這種情況下，參與者創建*威脅*事件，並透過該事件以特定效果（例如竊取資料或導致功能行為不當）來威脅系統。

功能和資料的結合，在系統中創造了**價值**，而造成威脅的對手則否定了該價值，這構成**風險**的基礎。風險會被控制所抵消，控制指的是涵蓋系統的功能、設計和構建系統的團隊組織行為，以及其成員對系統的操作，但這是有機率被修改的——攻擊者期望造成對系統的傷害，以及他們企圖嘗試的行為能夠成功。

每個概念和術語都需要進一步解釋才能有意義：

弱點

弱點是一種潛在的缺陷，它會修改系統行為或系統功能（導致不正確的行為），或者允許對資料進行未經驗證或不正確的存取。系統設計中的弱點是源於未能遵循最佳實踐、標準或約定俗成的規範而導致的，並且會對系統造成一些不良影響。對於威脅建模者（和開發團隊）來說，幸運的是，有一項社群倡議——常見弱點枚舉（CWE）（*http://cwe.mitre.org*）——已經創建了一個開放的安全弱點分類法，可以在調查系統設計問題時參考。

可利用性

可利用性是衡量攻擊者利用弱點造成傷害的難易程度。換句話說，可利用性是弱點對外部影響的暴露程度[12]。

漏洞

當一個弱點是可利用的（本地端所需授權的環境之外，其可利用性不為零），即被稱之為漏洞。漏洞為懷有惡意的對手提供了一種對系統造成某種破壞的手段。系統中既存但是在以前尚未被發現的漏洞，通常將其稱為**零日漏洞**。零日漏洞與其他類似性質的漏洞一樣危險，但它們之所以比較特殊，是因為很可能尚未找到解決方法來處理它們，因此被利用的可能性可能會增加。與弱點一樣，社區努力創建了漏洞分類法，並將所發現的漏洞編碼在 CVE（*https://cve.mitre.org*）資料庫中。

嚴重性

弱點會對系統及其資產（功能和資料）產生影響；這種問題的潛在損害和「爆炸半徑」被描述為缺陷的嚴重程度。對於那些主要職業是工程師或曾經在任何工程領域的人來說，嚴重性可能是一個熟悉的術語。根據定義，漏洞（可利用的弱點）至少與潛在的缺陷一樣嚴重，而且缺陷的嚴重性通常會更高，因為它很容易被利用。第 xxx 頁的「計算嚴重性或風險」中描述了計算嚴重性的方法。

12 在這裡使用的「外部」一詞是相對的，並且特定於所謂需要授權的環境；例如，作業系統、應用程式、資料庫等。

不幸的是，確定弱點嚴重程度的過程並不總是那麼乾脆。如果在發現弱點的同時，未能辨識出利用缺陷以造成影響的能力，那麼問題會有多麼嚴重？如果後來該缺陷被確認為已經暴露，或者更糟的是，該缺陷是源於系統設計或實作面的變化而暴露，那又會發生什麼情況？這些都是很難回答的問題。稍後我們將在介紹風險概念時談到這一點。

影響

如果弱點或漏洞被利用，則會對系統造成某種影響，例如破壞系統功能或暴露資料。在對問題的嚴重性進行分級，而你需要評估其影響級別時，須以漏洞被成功地利用進而造成的功能和資料潛在損失，作為衡量標準。

參與者

在描述系統時，參與者是指與系統相關的任何個體，例如用戶或攻擊者。具有惡意的行為者，無論是組織內部還是外部，創建或使用系統的，有時亦被稱為**對手**。

威脅

威脅是攻擊者以非零的機率來利用漏洞，以特定方式對系統造成負面影響的結果（通常用「對……的威脅」或「……的威脅」來表達）。

威脅事件

當對手試圖（成功或不成功）利用漏洞達到預期目標或結果時，這被稱為威脅事件。

損失

就本書的目的和威脅建模的主題而言，當對手造成威脅事件以一個（或多個）影響功能和數據時，就會發生損失：

- 攻擊者能夠破壞系統資料的機密性，以洩露敏感或私人的資訊。
- 攻擊者可以修改功能介面、更改功能行為或更改資料的內容或來源。
- 攻擊者可以暫時或永久地阻止已被授權的實體訪問系統功能或資料。

此處所指的損失以資產或具備價值的數量來描述。

風險

風險將目標被攻擊的潛在價值與實現影響的可能性相結合，不同於系統擁有人或資訊持有者評估價值的角度，這個潛在價值是取決於攻擊者的。你應該使用風險來告知問題的優先等級，並決定是否解決該問題。應優先考慮緩解那些易於被利用的嚴重漏洞，以及可能會導致重大損失的漏洞。

計算嚴重性或風險

我們可將嚴重性（成功地利用漏洞可能造成的損害程度）和風險（發起威脅事件的可能性與成功利用漏洞以產生負面影響的可能性的組合）公式化，以便於計算評估。雖然這些公式並不完美，但使用它們可以提供**一致性**。儘管現今存在許多確認嚴重性或風險的方法，但其中有一些威脅建模方法選擇使用本書未加以描述的風險評分方式。此處提供了三種常用方法的範例（一種用於測量嚴重性，兩種用於風險）。

CVSS（嚴重性）

通用漏洞評分系統（CVSS）（*https://www.first.org/cvss*）現在是 3.1 版，是安全事件應變和安全團隊論壇（FIRST）的產品。

CVSS 是一種數值量化方法，藉由建立數值從 0.0 到 10.0，它允許你識別**嚴重性**的組成部分。該計算基於成功利用漏洞的可能性以及對潛在影響（或損害）的衡量。如圖 I-3 所示，在計算器中設置了八個指標或數值來得出嚴重性等級。

成功的可能性是根據特定指標來衡量的，並且會給出對應的數字等級。這會產生一個稱為**可利用性子分數**的數值。以類似的方式（使用不同的指標）衡量影響性，則稱為**影響性子分數**。將這兩個分數加在一起，得出一個加總的**基本分數**。

請記住，CVSS 衡量的不是風險而是**嚴重程度**。CVSS 可以告訴你攻擊者成功利用受影響系統的漏洞可能性，以及它們可以造成的破壞程度。但它**無法**表明攻擊者何時或攻擊者是否會嘗試利用該漏洞；它也不能告訴你受影響資源，其價值多少或解決漏洞的成本。它只是針對發起攻擊的可能性、系統或功能的價值以及緩解它的成本，以此推動風險計算。依靠原始的嚴重性是傳達有關缺陷資訊的好方法，但這是一種非常不完善的風險管理方法。

可被利用性指標		
攻擊向量	AV	**網絡** 鄰近的 本地端的 實體的
攻擊複雜度	AC	**低** 高
需要特權	PR	**無** **低** 高
使用者互動	UI	**無** 需要

影響指標		
範圍變更	SC	**有變更** 無變更
機密性	C	**無** **低** 高
完整性	I	**無** **低** 高
可用性	A	**無** **低** 高

圖 I-3　通用漏洞評分系統指標、向量和分數

DREAD（風險）

DREAD 是一種古老但具基礎性的重要方法[13]，用於了解安全問題帶來的風險。DREAD 是 STRIDE 威脅建模方法的合作夥伴；我們將在第 3 章深入地討論 STRIDE。

DREAD 是以下各項的首字母縮寫詞：

損害（*Damage*）

如果對手發動攻擊，他們會造成多大的破壞？

可再現性（*Reproducibility*）

潛在攻擊是否容易被重現（在方法和效果上）？

可利用性（*Exploitability*）

進行一次成功的攻擊有多容易？

13 有人說 DREAD 已經失去了它的用處；參見 Irene Michlin，「威脅優先排序：恐懼已死，寶貝？」，NCC Group，2016 年 3 月，*https://oreil.ly/SJnsR*。