DOI 10.34185/1562-9945-3-146-2023-03
УДК 004.056

О.Г. Трофименко, А.І. Дика, Ю.Г. Лобода

АНАЛІЗ УРАЗЛИВОСТЕЙ ТА ПРОБЛЕМ БЕЗПЕКИ ВЕБЗАСТОСУНКІВ

Анотація. Поширеність порушень безпеки сайтів та вебзастосунків зробило тестування безпеки невіддільною складовою життєвого циклу розробки відповідного ПЗ. Пошук ефективних шляхів щодо мінімізації ризиків та вразливостей безпеки вебзастосунків є вельми актуальним. У статті проведено комплексний аналіз уразливостей, методів, інструментів та проблем, з якими стикається тестування безпеки вебзастосунків. З'ясовано, що до списку найпоширеніших уразливостей вебзастосунків увійшли: порушений контроль доступу, криптографічні збої, неправильна конфігурація безпеки, SQL та іншого роду ін'єкції, незахищений дизайн, помилки ідентифікації й автентифікації тощо. Виділено особливості специфіки уразливостей безпеки вебзастосунків. Окремо розглянуто проблеми, з якими стикаються автоматизовані інструменти для тестування веббезпеки.
Ключові слова: тестування безпеки, вебзастосунок, безпека вебзастосунків, уразливості безпеки, інструменти тестування.

Постановка проблеми. Сьогодні наявність стабільного і безпечного доступу до інтернет-ресурсів став життєво важливим для більшості нашого соціуму. Це стосується можливості працівників широкого спектра професій віддалено виконувати свої службові обов'язки, учнів та студентів – дистанційно навчатися і мати доступ до навчальних матеріалів, споживачів у всьому світі – вибирати онлайн товари та послуги засобами того чи іншого сайту чи то вебзастосунку.

Нині за умов розповсюдження кіберзлочинів та проведення цілеспрямованих інформаційних війн вебресурси повсюдно піддаються атакам різного масштабу і складності з метою впливу на конфіденційність, цілісність і доступність даних інформаційних систем. Поширеність порушень безпеки сайтів та вебзастосунків, а також важливість їх запобігання зробило тестування безпеки невіддільною складовою життєвого циклу розробки відповідного ПЗ, що має виявляти вразливості, пов'язані із забезпеченням цілісного підходу до захисту програми від хакерських атак, вірусів, несанкціонованого доступу до конфіденційних даних.

Тестування веббезпеки перевіряє бізнес-логіку, проблеми автентифікації та авторизації, кодування вхідних і вихідних даних, а також інші можливі ризики, щоб виявити уразливі місця в безпеці і переконатися, що всі функції вебзастосунків безпечні. Для тестування безпеки важливо враховувати різницю між ризиком та ефективністю розроблених тестів безпеки щодо пом'якшення відповідного ризику безпеки для аналізу рентабельності інвестицій у безпеку [1].

Аналіз останніх досліджень і публікацій. Аналізу проблем тестування, усунення помилок та забезпечення безпеки вебпрограм присвячені численні дослідницькі публікації. У дослідженні [2] проаналізовано сукупність знань 80-ти технічних статей, пов'язаних із тестуванням безпеки вебзастосунків, опублікованих з 2005 по 2020 рік. У роботі [3] виявлено 69 проблем, з якими стикаються розробники та користувачі вебпрограм, запропоновано ієрархічну структуру для визначення та категоризації таких проблем, проте не з'ясовані специфіка та проблеми тестування безпеки. Автори статті [4] стверджують, що тестування залежить від технологій впровадження, тому методи тестування мають відстежувати всі проблеми при розробленні стратегії тестування та адаптуватися до динамічно поширеної природи Інтернету. У роботі [5] запропоновано таксономію методів тестування безпеки з 3-х основних категорій (ідентифікація, тестування та звітність) і 17-ти підкатегорій, що складаються з конкретних методів тестування безпеки (наприклад, тестування чорної, білої та сірої скриньок, оцінка ризиків тощо). У роботі [6] наголошено на тому, що для забезпечення безпеки вебсайту потрібен аналіз його уразливостей, відповідно до стандартизації безпеки Open Web Application Security Project (OWASP). Проведений аналіз публікацій виявив значну зацікавленість у пошуку ефективних шляхів щодо мінімізації ризиків та вразливостей безпеки сайтів.

Мета дослідження: проаналізувати й узагальнити уразливості та проблеми, з якими стикається тестування безпеки сайтів і вебзастосунків.

Викладення основного матеріалу дослідження.

1. Аналіз уразливостей безпеки вебзастосунків. Для проведення тестування безпеки вебзастосунків тестувальник безпеки (Pentester або AppSec Engineer), окрім впевненого володіння декількома мовами програмування зі списку найпоширеніших фреймворків, має розбиратися в протоколах передачі даних, знати операційні системи Windows/Unix на рівні адміністратора, а також практики DevSecOps/AppSec з різними підходами до їх застосування, вміти налаштовувати та адмініструвати поширені інструменти CI/CD, мати навички

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

скриптингу для автоматизації завдань. Але головним для таких фахівців є знання векторів атак (на різних рівнях), методів обходу захисту на рівні застосунків та засобів їх захисту, володіння прийомами виявлення та усунення поширених уразливостей, вміння роботи з інструментами аналізу та оцінки уразливостей і рівня захищеності. Відтак фахові вимоги до рівня компетентностей Pentester дуже високі, особливо, якщо враховувати необхідність постійного саморозвитку і відстеження появи нових інструментів тестування безпеки та вивчення оновлень функціоналу у наявних. Pentester має перевіряти стійкість захисту ПЗ від різних невизначених атак, адже будь-яка уразливість може стати потенційно критичною загрозою для працездатності програми.

Щоб полегшити роботу фахівців з тестування безпеки, відкрита некомерційна організація з безпеки програмного забезпечення в Інтернеті OWASP (Open Web Application Security Project) досліджує та кожні кілька років оновлює список найпопулярніших уразливостей вебзастосунків. 2021 року до цього списку увійшли [7]: порушений контроль доступу, криптографічні збої, неправильна конфігурація безпеки, SQL та іншого роду ін'єкції, незахищений дизайн, помилки ідентифікації та автентифікації тощо.

Аналіз специфіки найпоширеніших нині уразливостей безпеки вебзастосунків показав такі їхні особливості:

1) *порушений контроль доступу* нині є доволі поширеною уразливістю безпеки вебзастосунків. Збої контролю доступу зазвичай призводять до несанкціонованого розголошення інформації, модифікації чи знищення даних або виконання бізнес-функцій за межами обмежень користувача. Найбільше випадків зламу контролю доступу відбувається через [8]: порушення принципу привілеїв користувачів, обхід перевірок контролю доступу через змінення URL-адреси, дозвіл на перегляд або редагування стороннього облікового запису шляхом надання його унікального ідентифікатора (незахищені прямі посилання на об’єкти), доступ до API із відсутніми елементами керування доступом для POST, PUT і DELETE, маніпуляції метаданими, файлами cookie або прихованими полями для підвищення привілеїв, неправильна конфігурація CORS, що дозволяє отримати доступ до API з неавторизованих/ненадійних джерел, тощо;

2) *криптографічні збої* є головною причиною розкриття конфіденційних даних під час передавання і зберігання. Так, паролі, номери кредитних карток, медичні записи, особиста інформація та комерційна таємниця вимагають додаткового захисту, головним чином, позаяк ці дані підпадають під дію законів про конфіденційність, наприклад, Закону України «Про захист персональних

даних», Загального регламенту про захист персональних даних у межах Європейського Союзу (GDPR) чи то інших нормативних актів і стандартів, наприклад, стандарту безпеки даних індустрії міжнародних платіжних систем (Payment Card Industry Data Security Standard, PCI DSS). Тут важливими чинниками уразливостей є [9]: небезпека зовнішнього інтернет-трафіка, застарілі або слабкі криптографічні алгоритми, методи та протоколи, усталено згенеровані або повторно використані криптоключі, застарілі чи некриптографічні хеш-функції тощо;

3) *ін'єкція* – лідер із найнебезпечніших методів злому, метою яких переважно є отримання даних користувачів, зокрема логінів і паролів, які дадуть змогу авторизуватися у системі з правами якогось користувача. Найпоширенішим видом ін'єкцій є SQL-ін'єкція (SQLi), яка використовує наявні уразливості сайту для відправки шкідливого коду у запитах до сервера бази даних (БД). Для SQL-ін'єкції можуть бути використані будь-які вхідні дані сайту: логін, пароль, ключові слова для пошуку, теги введення, рядки запиту, файли cookie тощо. Іншими видами ін'єкцій є: NoSQL-ін'єкції, міжсайтові сценарії (Cross-Site Scripting, XSS), виконання команд операційної системи (ОС), LDAP-ін'єкції (Lightweight Directory Access Protocol), ін'єкції шаблонів (EL, Expression Language), ін'єкції виразів OGNL (Object Graph Navigation Library), ін'єкції XML, XPATH тощо [4]. Так, наприклад, атаки ін'єкцій EL є серйозними вразливостями, які дозволяють зловмисникам витягувати фрагменти інформації (маркери сеансу) або виконувати команди на віддаленому сервері. Тому важливо: шифрувати паролі перед зберіганням їх у БД, не використовувати дані користувача для побудови виразів (створення шаблонів), завжди перевіряти дані, введені користувачем, на боці сервера, а не вставляти їх одразу у запит до БД, і при цьому не обмежуватись перевірками лише на клієнтській стороні. Для запобігання ризиків ін'єкцій доцільний ретельний аналіз вихідного коду та застосування інструментів тестування для виявлення вразливостей до ін'єкцій. Автоматизоване тестування всіх параметрів (заголовків, URL-адрес, файлів cookie, даних JSON, SOAP і XML) різнобічними інструментами тестування безпеки, включаючи статичні (SAST), динамічні (DAST) та інтерактивні (IAST), допомагає виявити введені недоліки ін'єкцій перед розгортанням у робочому середовищі;

4) *незахищений дизайн* – відносно нова категорія ризиків безпеки вебзастосунків, пов'язана з популяризацією використання шаблонів проєктування та еталонних архітектур. Ця уразливість стосується відсутності або

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

неефективності дизайну контролю та недоліків архітектури вебзастосунків, серед яких поширеними є: незахищене або недостатньо захищене зберігання облікових даних, порушення межі довіри, відсутність захисту від ботів, створення повідомлень про помилки, що містять конфіденційну інформацію, тощо. Можливим наслідком використання цієї вразливості може бути, наприклад, спроможність моделювання зловмисниками потоку запитів на групове псевдобронювання квитків або неавтентичне замовлення товарів і, відповідно, значні проблеми і матеріальні збитки. Для запобігання подібного доцільно ретельно підходити до дизайну антиботів і правил логіки домену, варто інтегрувати перевірки правдоподібності на кожному рівні програми (від інтерфейсу до бекенду), слід обмежити користувачам і сервісам доступ до ресурсів, потрібно використовувати бібліотеки безпечних шаблонів проєктування і компонентів, застосовувати засоби контролю безпеки та конфіденційності, використовувати моделювання загроз для критичної автентифікації, контролю доступу, бізнес-логіки та потоків ключів, розробити модульні та інтеграційні тести для перевірки стійкості усіх критичних потоків до моделі загроз і сценаріїв атак;

5) *неправильна конфігурація безпеки* – уразливість, яка стає все більш поширеною через популярність ПЗ з широкими можливостями налаштування. Так, програма може бути уразливою, якщо: неправильно налаштовані дозволи для хмарних служб (наприклад, стандартні дозволи на спільний доступ до Інтернету для інших користувачів хмарних послуг дозволяють отримати доступ до конфіденційних даних, які зберігаються в хмарному сховищі), не використовується сегментована архітектура програм, облікові записи і паролі залишаються усталено активними та не змінюються, відсутній автоматизований процес перевірки ефективності конфігурацій і налаштувань у всіх середовищах, встановлено непотрібний функціонал чи то компоненти (наприклад, непотрібні порти, служби, сторінки, облікові записи або привілеї) і, навпаки, не налаштовано або взагалі вимкнено оновлення функцій безпеки для обновлюваних систем, використовується застаріле, вразливе ПЗ або не встановлено безпечні значення параметрів конфігурації безпеки для серверів, фреймворків, бібліотек, БД тощо;

6) *вразливі та застарілі компоненти* можуть призвести до серйозних наслідків як випадкових (наприклад, помилки кодування), так і навмисних (наприклад, бекдор у компоненті, віддалене виконання коду на сервері). Існують автоматизовані інструменти, які допомагають зловмисникам виявляти невиправлені або неправильно налаштовані системи. Тому важливо забезпечи-

ти постійний моніторинг та автоматизоване сканування всіх компонентів (як на боці клієнта, так і на боці сервера) у використовуваних середовищах розробки та своєчасне оновлення конфігурації ПЗ, включаючи ОС, вебсервери, системи керування базами даних (СКБД), API, бібліотеки та всі програмні компоненти середовища виконання. Крім того, варто одразу видаляти невикористовувані компоненти (фреймворки, бібліотеки) та їхні залежності, функції, файли та документацію. Щоб зменшити ймовірність долучення модифікованих шкідливих компонентів (як клієнтських, так і серверних), рекомендовано завантажувати їх лише з офіційних джерел через безпечні посилання та підписатися на сповіщення про вразливості безпеки використовуваних компонентів;

7) *помилки ідентифікації та автентифікації* є поширеними для сценаріїв атак. Уразливості автентифікації ПЗ можуть бути викликані різними причинами, серед яких: дозвіл задавати слабкі або поширено відомі паролі; використання неефективного відновлення облікових даних і забутих паролів; відсутність шифрування або слабо хешовані сховища даних паролів; перекидання облікових даних; відсутність або неефективність багатофакторної автентифікації; повторне використання ідентифікатора сеансу після успішного входу; відсутність анулювання маркерів автентифікації після виходу із системи або після періоду бездіяльності тощо. Гарними практиками запобігання атак через помилки ідентифікації та автентифікації є: вимоги до ротації паролів і їх складності; застосування багатофакторної автентифікації задля запобігання автоматизованого перебору облікових даних, грубого форсування та атак повторного використання викрадених облікових даних; використання захищеного вбудованого менеджера сеансів на боці сервера, який генерує випадковий ідентифікатор сеансу з високою ентропією та анулює його після виходу із системи чи то тривалого простою; обмеження для невдалих спроб входу; реєстрація збоїв та сповіщення адміністраторів у разі виявлення перекидання облікових даних чи то інших атак;

8) *порушення цілісності програмного забезпечення та даних* – відносно нова категорія уразливостей, яка стосується коду або інфраструктури, що долучаються без перевірки їх цілісності. Прикладом цього може бути ситуація, коли програма використовує плагіни, бібліотеки або модулі з ненадійних джерел. Так, нині поширено ПЗ використовує автоматичне оновлення і при цьому оновлення завантажуються без достатньої перевірки цілісності. Тобто потенційно зловмисники можуть завантажити власні оновлення для розпов-

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

сюдження функціональності з ненадійної сфери управління та запуску на всіх інсталяціях. Для запобігання таких ризиків варто: використовувати надійні репозиторії бібліотек, перевіряти код та зміни конфігурації для мінімізації ймовірності потрапляння зловмисного коду у конвеєр ПЗ, використовувати інструменти безпеки ланцюга поставок ПЗ для перевірки компонентів на наявність відомих уразливостей;

9) *збої в журналі безпеки та моніторингу* – уразливість, яку важко перевірити, але вона може спричинити злом і доступ до конфіденційних даних клієнтів ПЗ, наприклад: медичних записів, даних паспортів і кредитних карток, платіжних записів клієнтів, і, як наслідок, до моральних і матеріальних збитків. Відсутність моніторингу та реєстрації вебзастосунка позбавляє можливості виявити злом системи і витоки даних. З іншого боку, навіть із журналом моніторингу доволі проблематично виявити атаки на проникнення, адже навіть тестування на проникнення і сканування інструментами динамічного тестування безпеки застосунків не завжди формують відповідні попередження. Так, програма може не виявляти або не сповіщати про активні атаки в режимі реального часу для оперативного реагування, наприклад, коли журнали зберігаються лише локально або відповідні порогові значення сповіщень відсутні чи то не діють. Тому розробники мають переконатися, що всі помилки входу, контролю доступу та перевірки введення на боці сервера реєструються і зберігаються для ідентифікації підозрілих або зловмисних облікових записів. При цьому дані журналу, з одного боку, мають бути закодовані, щоб запобігти ін'єкціям або атакам на системи реєстрації чи моніторингу, а з іншого, журнали доцільно генерувати у форматі, який можна легко використовувати для ефективного моніторингу, контролю цілісності та оперативного формування попереджень про підозрілі дії для швидкого реагування на них;

10) *підробка запитів на боці сервера* (Server-Side Request Forgery, SSRF) не є дуже поширеною уразливістю, проте саме вона дозволяє зловмисникам надсилати URL-запити від імені скомпрометованого сервера в обхід контролю доступу до мережі. Подібного роду атаки можуть стосуватися проникнення через відкриті порти внутрішніх серверів і доступ до конфіденційних даних через сховища метаданих хмарних служб, локальні файли або внутрішні служби. Помилки цієї категорії виникають, коли вебзастосунок отримує віддалений ресурс без перевірки наданої користувачем URL-адреси. Це надає зловмиснику можливість частково або повністю взяти під контроль сервер, щоб змусити його виконувати запити віддалено, навіть якщо він захищений мережевим екра-

ном, VPN чи то іншим типом контролю доступу до мережі. А оскільки сучасні вебзастосунки нині масово надають кінцевим користувачам зручні функції, тому отримання URL-адреси від користувача стає звичною справою. Як наслідок, поширеність уразливості SSRF надалі набиратиме обертів. Крім того, створення ефективного захисту від таких атак ускладнюється через стрімкий розвиток хмарних послуг та ріст складності архітектури хмарних сервісів. Запобігти SSRF можна лише через комплекс засобів захисту в глибинних елементах керування, в який серед інших входить вимкнення перенаправлення HTTP та налаштування у брандмауері правил контролю доступу до мережі, щоб блокувати увесь інтрамережевий трафік, крім основного, та реєструвати всі прийняті і заблоковані мережеві потоки. Також розробники мають: заборонити надсилати необроблені відповіді клієнтам, очищати і перевіряти всі вхідні дані, надані клієнтом, не розгортати служби безпеки на зовнішніх системах і контролювати локальний трафік у цих системах, а для інтерфейсів із виділеними та керованими групами користувачів використовувати мережеве шифрування (наприклад, VPN) у незалежних системах, щоб враховувати високі потреби захисту.

Усі розглянуті види уразливостей вебресурсів безпосередньо пов’язані з відповідними ризиками тестування безпеки вебзастосунків і вебслужб.

2. Аналіз проблем вебтестування безпеки. Тестування безпеки вебпрограм допомагає імітувати та виявляти можливі уразливості та загрози, пов’язані з цією програмою. Воно перевіряє відповідність програми вимогам безпеки під час впливу будь-яких шкідливих вхідних даних.

Можливими проблемами, пов’язаними з аспектом вебтестування безпеки, є: зламані або ненадійні паролі, переповнення буфера, маніпулювання прихованими полями, ненадійне використання криптографії, перехоплення файлів cookie, неправильні конфігурації сервера, слабке керування сеансами, розкриття конфіденційних даних, маніпуляції з параметрами, соціальне хакерство, неадекватна перевірка введених даних тощо [10].

Проблеми, з якими стикається вебтестування безпеки, стосуються розробки автоматизованих інструментів для тестування веббезпеки, використання вебзастосунків RIA (Rich Internet Application), застосування незахищеного криптографічного сховища тощо [11]. Pentester має перевірити стійкість захисту ПЗ від різних невизначених атак, наприклад, від атак на відмову в обслуговуванні (DDoS attack), адже будь-яка уразливість може стати потенційно критичною загрозою для програми. Ризики безпеки вебзастосунку

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

можуть бути пов'язані з фазою проєктування, фазою розробки, фазою розгортання, фазою технічного обслуговування. Саме тестування безпеки використовується для виявлення цих ризиків вебзастосунку, дослідження уразливостей і слабких місць вебпрограми. Наприклад, під час SQL-ін'єкції зловмисник може змінювати SQL-запити і в такий спосіб отримати неавторизований доступ до серверної частини. Ризик, пов'язаний із цією уразливістю, може полягати в тому, що зловмисник зможе отримати доступ до даних, на які він не авторизований.

Розробка автоматизованих інструментів завжди була проблемою тестування вебзастосунків на безпеку. Створити автоматизовані інструменти для тестування безпеки важче, ніж для тестування функціональності вебзастосунку [12]. Традиційні інструменти не встигають за темпами екосистеми програмного забезпечення. Проблеми, з якими стикаються автоматизовані інструменти для тестування веббезпеки, полягають у тому, що вони повинні йти в ногу з технологіями, що стрімко розвиваються та змінюються (наприклад, RIA), і належним чином інтегруватися в наявні робочі процеси розробки.

Зосередження на різних питаннях і проблемах, пов'язаних із тестуванням безпеки вебзастосунків, надає значні дивіденди у виявленні та усуненні різноманітних ризиків, уразливостей, атак, загроз, вірусів тощо. Крім того, корисно навчити тестувальника безпеки вміло моделювати тестову програму та розробляти відповідну стратегію тестування. Важливо розуміти і оцінювати рівень безпеки, який вимагатиме певний проєкт. Активи, які підлягають захисту, слід класифікувати і зазначенням їх рівня, наприклад: конфіденційно, секретно, цілком таємно. Також важливо переконатися, що усі законодавчі вимоги безпеки будуть дотримані у вебзастосунку та його оточенні, базуючись на принципах конфіденційності, доступності та цілісності при зберіганні даних облікових записів, доступі і підключеннях користувачів. Управління ризиками безпеки й відповідність нормативним вимогам належать до політик і процесів, які організації мають забезпечити для дотримання законів, правил та нормативних актів.

Висновки. Проведений аналіз уразливостей безпеки, методів та проблем тестування вебзастосунків виявив наявність різних підходів для захисту програмних продуктів. Доволі не просто зорієнтуватися і зрозуміти, які саме методи використовувати або коли і в якій послідовності застосовувати ті чи інші засоби тестування. Доцільним є баланс у використанні декількох різних методів і підходів, які будуть доповнювати і посилювати один одного.

Комплексний підхід має інтегрувати тестування у всі етапи життєвого циклу розробки ПЗ. Саме такий підхід допомагає використовувати найбільш відповідні й ефективні доступні методи для поточної фази розробки програмного продукту. Ручне й автоматизоване тестування безпеки вебзастосунків мають доповнювати одне одного, оскільки обидва вони є однаково важливими процесами і мають одну мету – захистити програму. Доцільним є поєднання обох методів, починаючи з автоматизованого тестування безпеки та доповнюючи його ручним тестуванням на проникнення. Звісно, бувають виняткові ситуації та обставини, коли доречна лише якась одна техніка. Наприклад, якщо тестувальник не має доступу до вихідного коду, а є потреба тестування на проникнення, адже і таке тестування вочевидь краще, аніж відсутність тестування взагалі. Звичайно таких ситуацій краще уникати і виконувати повноцінно якісне тестування безпеки. А тому існує реальна потреба створювати ефективне тестове середовище для проведення тестування безпеки вебзастосунків.

Збалансований підхід до тестування безпеки залежить від багатьох чинників, у тому числі і зрілості процесу тестування та корпоративної культури, але важливо, щоб компанії приділяли увагу інтеграції тестування безпеки на якомога ранніх етапах життєвого циклу розробки ПЗ. Відповідальність за визначення та реалізацію ефективних заходів безпеки для свого процесу нині перейшла до самих розробників. Тестування безпеки має поєднувати різні ефективні техніки і методології для оцінки безпеки програми та її оточення. Адже жодна технологія не захищена від хакерів чи кіберзлочинців, а потреба у захисті різних типів вебзастосунків нині, як ніколи, актуальна.

ЛІТЕРАТУРА

1. Web Security Testing Guide. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/stable/2-Introduction/

2. Aydos M. Security testing of web applications: A systematic mapping of the literature / Aydos M., Aldan Ç., Coşkun E., Soydan A. // Journal of King Saud University - Computer and Information Sciences. – 2022. – Vol. 34, Issue 9. – P. 6775-6792. DOI: 10.1016/j.jksuci.2021.09.018.

3. Mubshra Q. A Rigorous Approach to Prioritizing Challenges of Web-Based Application Systems / Mubshra Q., Shahid F., Mohd H., Nizam B., Md N., Atif A. // Malaysian Journal of Computer Science. – 2021. – № 34. DOI: 10.22452/mjcs.vol34no2.1.

4. Arunima J. Security Testing of Web Applications: Issues and Challenges / Arunima J., Gaurav R., Dheerendra S. // International Journal of Computer Applications. – 2014. – № 88. DOI: 10.5120/15334-3667.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

5. Omer T. Analysis of Security Testing Techniques. / Omer T., Sadeeq J., Alaa K., Adil K., Fazal K., Sana K. // Intelligent Automation and Soft Computing. – 2021. – № 29. – P. 291-306. DOI: 10.32604/iasc.2021.017260.
6. Riandhanu I. Analisis Metode Open Web Application Security Project (OWASP) Menggunakan Penetration Testing pada Keamanan Website Absensi. / Riandhanu I. // Jurnal Informasi dan Teknologi. – 2022. – Vol. 4, No. 3. – DOI: 10.37034/jidt.v4i3.236.
7. OWASP Top Ten [Online]. Available: https://owasp.org/www-project-top-ten
8. Broken Access Control – A01:2021 [Online]. Available: https://owasp.org/Top10/A01_2021-Broken_Access_Control/
9. Cryptographic Failures – A02:2021 [Online]. Available: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/#list-of-mapped-cwes
10. Web Security Testing Guide. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/stable/2-Introduction/
11. Keshav M. Automated VS Manual Security Testing – Which One to Choose? [Online]. Available: https://www.getastra.com/blog/security-audit/manual-security-testing/
12. Трофименко О.Г. Автоматизація тестування вебсайтів електронної комерції. / Трофименко О.Г., Пастернак Ю.Ю., Манаков С.Ю., Лобода Ю.Г. // Сучасна спеціальна техніка. – 2021. – № 2(65). – С. 46-59. DOI: 10.36486/mst2411–3816.2021.2(65).5.

REFERENCES

1. Web Security Testing Guide. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/stable/2-Introduction/
2. Aydos M. Security testing of web applications: A systematic mapping of the literature / Aydos M., Aldan Ç., Coşkun E., Soydan A. // Journal of King Saud University - Computer and Information Sciences. – 2022. – Vol. 34, Issue 9. – P. 6775-6792. DOI: 10.1016/j.jksuci.2021.09.018.
3. Mubshra Q. A Rigorous Approach to Prioritizing Challenges of Web-Based Application Systems / Mubshra Q., Shahid F., Mohd H., Nizam B., Md N., Atif A. // Malaysian Journal of Computer Science. – 2021. – № 34. DOI: 10.22452/mjcs.vol34no2.1.
4. Arunima J. Security Testing of Web Applications: Issues and Challenges / Arunima J., Gaurav R., Dheerendra S. // International Journal of Computer Applications. – 2014. – № 88. DOI: 10.5120/15334-3667.
5. Omer T. Analysis of Security Testing Techniques. / Omer T., Sadeeq J., Alaa K., Adil K., Fazal K., Sana K. // Intelligent Automation and Soft Computing. – 2021. – №

29. – P. 291-306. DOI: 10.32604/iasc.2021.017260.
6. Riandhanu I. Analisis Metode Open Web Application Security Project (OWASP) Menggunakan Penetration Testing pada Keamanan Website Absensi. / Riandhanu I. // Jurnal Informasi dan Teknologi. – 2022. – Vol. 4, No. 3. – DOI: 10.37034/jidt.v4i3.236.
7. OWASP Top Ten [Online]. Available: https://owasp.org/www-project-top-ten
8. Broken Access Control – A01:2021 [Online]. Available: https://owasp.org/Top10/A01_2021-Broken_Access_Control/
9. Cryptographic Failures – A02:2021 [Online]. Available: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/#list-of-mapped-cwes
10. Web Security Testing Guide. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/stable/2-Introduction/
11. Keshav M. Automated VS Manual Security Testing – Which One to Choose? [Online]. Available: https://www.getastra.com/blog/security-audit/manual-security-testing/
12. Trofymenko O. Automation of testing e-commerce websites / Trofymenko O., Pasternak Yu., Manakov S., Loboda Yu. // Modern Special Technics. – 2021. – № 2(65). – C. 46-59. DOI: 10.36486/mst2411–3816.2021.2(65).5.

Received 31.03.2023.
Accepted 03.04.2023.

Analysis of vulnerabilities and security problems of web applications

The article provides a comprehensive analysis of vulnerabilities, methods, tools and problems faced by web application security testing. The analysis of scientific research in the field of web application security testing revealed a significant interest of scientists in finding effective ways to minimize site security risks and vulnerabilities. It was found out that the list of the most common web application vulnerabilities includes: broken access control, cryptographic failures, misconfiguration of security, SQL and other injections, insecure design, identification and authentication errors, etc. Specific features of the security vulnerabilities of web applications are highlighted. The problems faced by automated tools for web security testing are separately considered, namely the development of automated tools for web security testing, the use of RIA (Rich Internet Application) web applications, and the use of insecure cryptographic storage. Web application security risks can be associated with the design phase, the development phase, the deployment phase, and the maintenance phase. It is security testing that is used to identify these risks of the web application, to investigate the vulnerabilities and weak points of the web application. The conducted analysis of security vulnerabilities,

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

methods and problems of testing web applications revealed the presence of different approaches to protect software products. A combination of manual and automated web application security testing techniques is advisable, starting with automated security testing and complementing it with manual penetration testing. A comprehensive approach should integrate testing into all stages of the software development life cycle. Such approach helps to use the most appropriate and effective available methods for the current phase of software product development.

Keywords: security testing, web application, web application security, security vulnerabilities, testing tools.

Трофименко Олена Григорівна - кандидат технічних наук, доцент, доцент кафедри інформаційних технологій Національного університету «Одеська юридична академія».

Дика Анастасія Іванівна - асистент кафедри інформаційних технологій Національного університету «Одеська юридична академія».

Лобода Юлія Геннадіївна - кандидат технічних наук, доцент, доцент кафедри інформаційних технологій Національного університету «Одеська юридична академія».

Trofymenko Olena - PhD, Associate Professor, Associate Professor at the Department of Information Technologies of the National University "Odessa Law Academy".

Dyka Anastasiia - Assistant of Department of Information Technologies of the National University “Odessa Academy of Law”

Loboda Yuliia - PhD, Associate Professor, Associate Professor at the Department of Information Technologies of the National University "Odessa Law Academy".