УДК 004.056
DOI: 10.31673/2409-7292.2022.023037

Пономаренко М.О., Запорожченко М.М.

РЕКОМЕНДАЦІЇ ЩОДО ПІДВИЩЕННЯ ЗАХИЩЕНОСТІ ВЕБ-РЕСУРСУ У ХМАРНОМУ СЕРЕДОВИЩІ AMAZON WEB SERVICE

У даній статті надаються відомості про структуру веб-ресурсу, методи розробки програмного забезпечення та відомі веб-атаки на веб-ресурси. Надано рекомендації щодо захисту від веб-атак, їх попередження та виявлення за допомогою впровадження методики безпечної розробки та використання інструментів виявлення вразливостей. Запропоновано рекомендації щодо захисту інфраструктури веб-ресурсу.

Ключові слова: веб-ресурс, веб-атака, вразливості веб-ресурсу, зловмисник, захищеність веб-ресурсу, цикл розробки, неперервна інтеграція, неперервна доставка, SAST, DAST, SCA.

Вступ

У зв’язку з глобальної діджиталізацією, все більше компаній вкладають гроші у розвиток власних веб-ресурсів. Деякі компанії, такі як Facebook, отримують прибуток лише від користувачів у інтернет просторі. Компанії, потужності яких знаходилися в своїх датацентрах, вдаються до використання хмарних технологій і переносу своїх продуктів у веб. Через те, що архітектура веб-ресурсів стає більш складною, використовується багато мов програмування, фреймворків та залежностей, хакери знаходять нові вразливості у програмному коді та використовують їх, що може призвести до великих економічних та репутаційних втрат. Тому питання безпеки стає ключовим у всьому життєвому циклі розробки програмного забезпечення: починаючи від визначення вимог безпеки до реалізації продукту у інтернеті. При правильному налаштуванні процесів, можна побудувати більш безпечний додаток, шляхом зменшення кількості і серйозності вразливостей, а також знизити вартість розробки.

Мета статті – розробити рекомендації щодо застосування методів та засобів протидії вразливостям веб-ресурсу у хмарному середовищі Amazon Web Service.

Хмарне середовище Amazon Web Service

Багато компаній, які розвиваються, поступово переходять до використання IaaS. Як зазначалося раніше, даний сервіс дає можливість орендувати комп’ютерні потужності, мережі, сховища даних, а також використовувати сервіси, які надаються платформою. Amazon Web Services являється яскравим прикладом IaaS.

За допомоги EC2 сервісу, в AWS можна отримати в користування віртуальні машини «інстанси», які можуть відрізнятися виділеною кількістю центральних процесорів та пам’яті. Зазвичай такі комп’ютерні потужності поставляються з допоміжними функціями, такими як автоматичне масштабування, балансування навантаження (LB), проксування (NAT Gateway).

Мережа в хмарі – це форма програмно-визначеної мережі, в якій традиційне мережеве обладнання, таке як маршрутизатори та комутатори, стає доступним програмно, зазвичай через API. Для створення і керування мережами у AWS існує сервіс VPC. В залежності від клієнтоорієнтованості веб-ресурсу, можна розробити мережі в різних регіонах, тобто в різних локаціях розміщення датацентрів, для забезпечення найкращого користувальницького досвіду.

В AWS існує три основні типи хмарних сховищ - це блочне сховище, сховище файлів та сховище об'єктів (S3). Блокові та файлові сховища поширені у традиційних центрах обробки даних, але їм часто не вистачає масштабу, продуктивності та розподілених характеристик хмари. Таким чином, об'єктне сховище стало найпоширенішим способом зберігання у хмарі, враховуючи його високий ступінь розподіленості (і, отже, відмовостійкість), використання загальнодоступного обладнання, доступ до даних можна легко отримати через HTTP, а масштабування – це не лише практично безмежне, але продуктивність лінійно зростає зі зростанням кластера.

На рисунку 1 зображено різницю між типами розгортання програмного забезпечення. AWS також пропонує використання сервісів контейнеризації, такі як Elastic Container

Services (ECS), Elastic Kubernetes Services (EKS), а також сховища для створених образів Elastic Container Registry (ECR), які використовуються для відновлення роботи системи, в разі невдалого оновлення веб-ресурсу. Більш того, дане хмарне середовище пропонує використовувати «serverless» рішення для контейнеризації, тобто забираючи процес створення і підтримки на свою сторону. Даний сервіс являється більш захищеним, оскільки AWS максимально ізолює контейнер і піклується про захист віртуальної машини, але менш контрольованим, через неможливість отримання доступу до ресурсу.

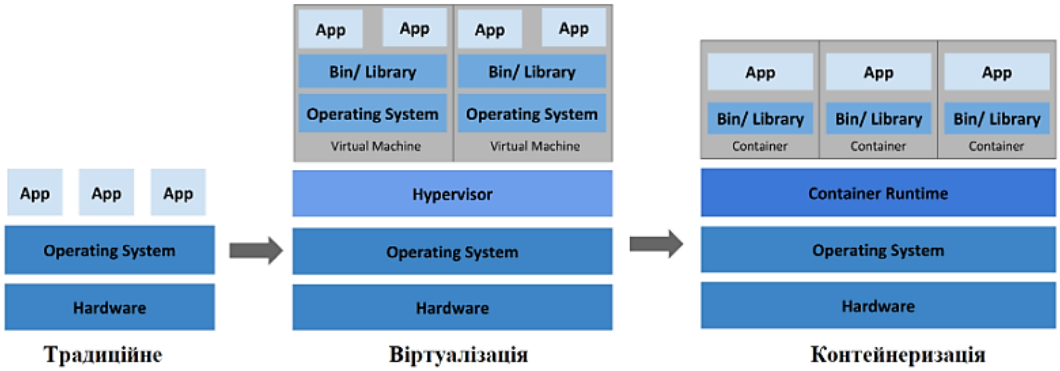

Рис. 1. Типи розгортання програмного забезпечення

У Amazon Web Services є певні недоліки, пов'язані з відсутністю центральної консолі використаних ресурсів, а також швидким керуванням даними ресурсами, які вирішуються за допомогою використання інфраструктури як код (IaC). Інфраструктура як код (IaC) – це управління інфраструктурою (мережами, віртуальними машинами, балансувальниками навантаження та топологією з'єднань) в описовій моделі з використанням того ж управління версіями, яке команда DevOps використовує для вихідного коду. Подібно до принципу, згідно з яким один і той же вихідний код генерує один і той же двійковий файл, модель IaC створює те саме середовище при кожному застосуванні. IaC – це ключовий метод DevOps, який використовується у поєднанні з безперервною доставкою. Основними перевагами IaC є:

прискорення виробництва та виведення товару на ринок;

стабільність середовища, усунення дрейфу конфігурацій, яке відбувається, коли довільні зміни та оновлення конфігурації призводять до розбіжності середовищ розробки, тестування та розгортання;

швидша та ефективна розробка.

Спрощуючи надання інфраструктури та підвищуючи її консистентність, IaC прискорює кожен етап життєвого циклу доставки ПЗ. Розробники можуть швидко підготувати «пісочниці» та середовища безперервної інтеграції/безперервного розгортання (CI/CD). Найшвидше надаються тестові середовища, інфраструктура для перевірки безпеки. Популярними продуктами для розробки IaC являються Terraform, Pulumi, AWS CloudFormation та інші.

Розгортання інфраструктури в хмарному середовищі AWS

Для виконання даної роботи було використано такі інструменти і сервіси:

AWS (Amazon Web Services): VPC, ECS, ECR, ALB, NAT Gateway, EC2, S3.

інфраструктура як інструмент коду (IaC): Terraform;

створення процесу неперервної інтеграції та доставки: Jenkins;

SCA: Dependency Check;

DAST: OWASP ZAP;

контейнеризація додатків: Docker;

система контролю версій: Github.

Першою чергою, необхідно розгорнути інфраструктуру в AWS за допомогою інструменту Terraform. На рисунку 2 можна побачити кількість ресурсів, які будуть

розгорнуті. На даному рисунку зображено інфраструктуру проекту. Переходячи на веб-ресурс, користувач звертається до балансувальника навантаження (ALB). Балансувальник перевіряє стан контейнерів і ділить трафік між екземплярами веб-ресурсів. Якщо стан контейнера незадовільний, то він буде видалений, а на його місце прийде інший. Трафік не буде перенаправлятися на новий екземпляр, доки балансувальник не впевниться в його готовності. Після обробки запиту, відповідь відправляється користувачу назад через механізм транслювання адрес NAT.

```
  # aws_vpc.test-vpc will be created
  + resource "aws_vpc" "test-vpc" {
      + arn                                  = (known after apply)
      + cidr_block                           = "172.16.0.0/16"
      + default_network_acl_id               = (known after apply)
      + default_route_table_id               = (known after apply)
      + default_security_group_id            = (known after apply)
      + dhcp_options_id                      = (known after apply)
      + enable_classiclink                   = (known after apply)
      + enable_classiclink_dns_support       = (known after apply)
      + enable_dns_hostnames                 = (known after apply)
      + enable_dns_support                   = true
      + id                                   = (known after apply)
      + instance_tenancy                     = "default"
      + ipv6_association_id                  = (known after apply)
      + ipv6_cidr_block                      = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + tags_all                             = (known after apply)
    }

Plan: 36 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + alb_hostname          = (known after apply)
  + aws_ecr_repository_url = (known after apply)
```

Рис. 2. Створення ресурсів у хмарному середовищі

Створені ресурси будуть мати наступну схему (рис. 3):

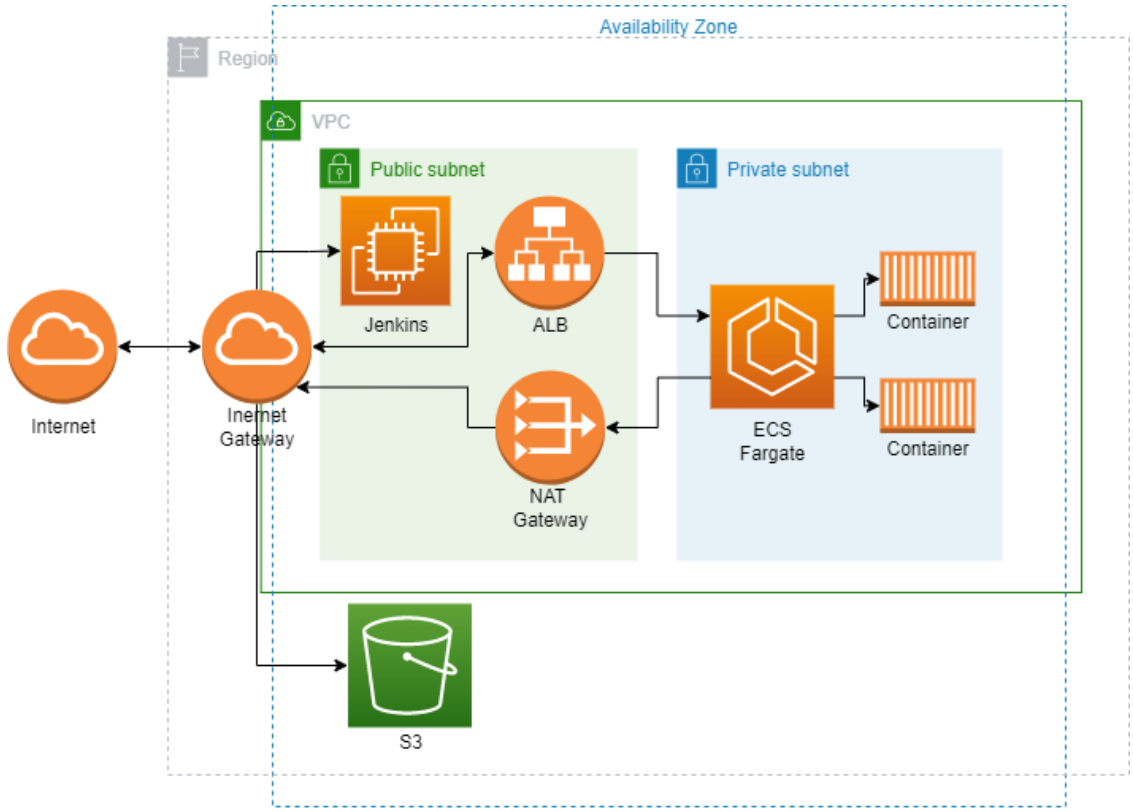

Рис. 3. Запуск інфраструктури за допомогою Terraform

Розгортання захищеного циклу неперервної інтеграції та доставки

Для створення життєвого циклу програмного забезпечення, було використано CI інструмент Jenkins, за допомогою якого ми будемо автоматизувати процеси перевірки безпеки, тестування та розгортання додатку. У моєму випадку було використано додаток на основі Java. Код програми міститься у репозиторії Github. Під час встановлення програма надає можливість встановити рекомендовані плагіни. Для налаштування роботи з AWS було інстальовано додаткові плагіни, як показано на рисунку 4.

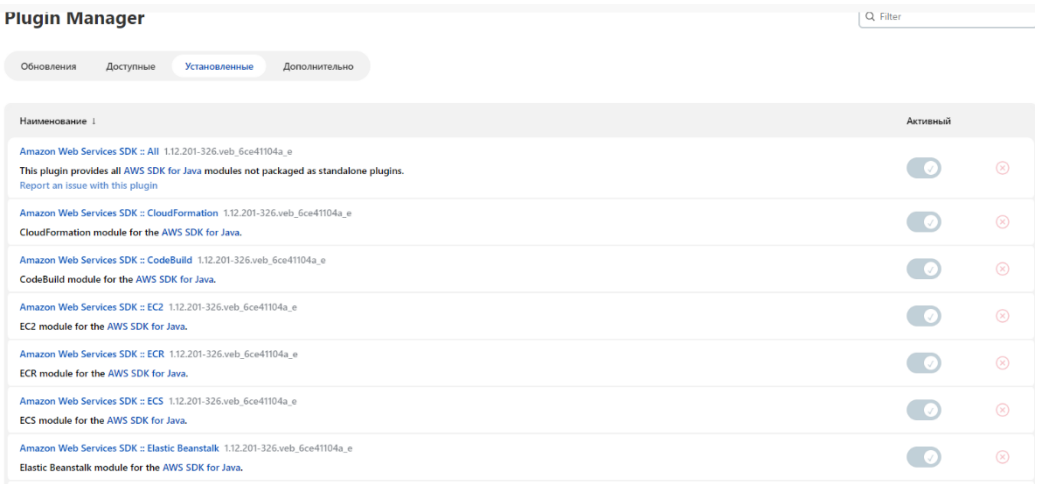

Рис. 4. Список використаних плагінів Jenkins

Також для автоматизації запуску пайплайнів в Jenkins, в залежності від гілки, необхідно налаштувати вебхук в Github, тобто, коли розробник вносить зміни до репозиторію, автоматично запускається пайплайн, який перевіряє код на наявність помилок. На рисунку 5 зображено, як саме налаштовується вебхук.

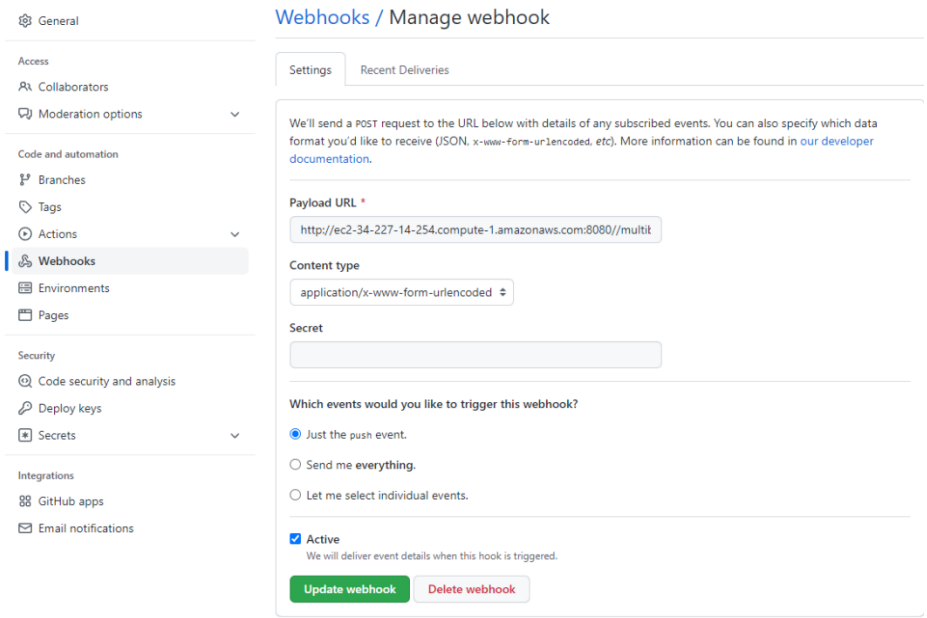

Рис. 5. Процес налаштування github webhook

Після налаштування Jenkins розробляється конфігурація життєвого циклу програмного забезпечення на мові groovy, пайплайн, в якому детально описується кожен крок для автоматизації циклу розробки. До його складу входять етапи клонування коду з github

репозиторія, початковий аналіз коду, компіляція, виконання контейнеру додатку, сканування DAST за допомогою OWASP ZAP і зберігання образу контейнера в довіреному репозиторії з подальшим використанням в процесі доставки (рис. 6).

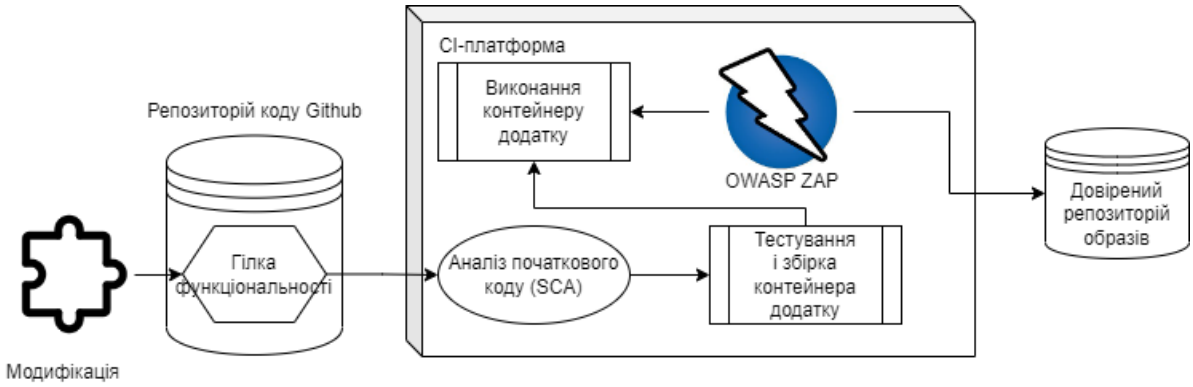

Рис. 6. Графічне зображення пайплайна CI

Для написання інструкцій використано декларативну форму мови groovy. Після перевірки коду на наявність вразливих бібліотек, проект збирається за допомогою Maven. У даному проекті Docker використовується для створення контейнерів додатку, а також для запуску сканерів Dependency Check та OWASP ZAP. Dependency-Check – це інструмент аналізу складу програмного забезпечення (SCA), який намагається виявити публічно розкриті вразливості, що містяться у залежностях проекту. Це досягається шляхом визначення наявності ідентифікатора Common Platform Enumeration (CPE) для цієї залежності. Якщо його знайдено, він створить звіт із посиланням на відповідні записи CVE. На рисунку 7 зображено виявлені вразливості, знайдені в залежностях нашого проекту.

postgresql-42.2.24.jar	cpe:2.3:a:postgresql:postgresql:42.2.24:*:*:*:*:*:*:* cpe:2.3:a:postgresql:postgresql_jdbc_driver:42.2.24:*:*:*:*:*:*:*	pkg:maven/org.postgresql/postgresql@42.2.24	CRITICAL	1
slf4j-api-1.7.32.jar		pkg:maven/org.slf4j/slf4j-api@1.7.32		0
snakeyaml-1.28.jar	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28:*:*:*:*:*:*:*	pkg:maven/org.yaml/snakeyaml@1.28		0
spring-boot-2.5.6.jar	cpe:2.3:a:vmware:spring_boot:2.5.6:*:*:*:*:*:*:*	pkg:maven/org.springframework.boot/spring-boot@2.5.6		0
spring-boot-devtools-2.5.6.jar: livereload.js				0
spring-core-5.3.12.jar	cpe:2.3:a:pivotal_software:spring_framework:5.3.12:*:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:5.3.12:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_framework:5.3.12:*:*:*:*:*:*:* cpe:2.3:a:vmware:springsource_spring_framework:5.3.12:*:*:*:*:*:*:*	pkg:maven/org.springframework/spring-core@5.3.12	CRITICAL	5
spring-data-commons-2.5.6.jar	cpe:2.3:a:pivotal_software:spring_data_commons:2.5.6:*:*:*:*:*:*:*	pkg:maven/org.springframework.data/spring-data-commons@2.5.6		0
spring-data-jpa-2.5.6.jar	cpe:2.3:a:pivotal_software:spring_data_jpa:2.5.6:*:*:*:*:*:*:*	pkg:maven/org.springframework.data/spring-data-jpa@2.5.6		0
spring-tx-5.3.12.jar	cpe:2.3:a:pivotal_software:spring_framework:5.3.12:*:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:5.3.12:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_framework:5.3.12:*:*:*:*:*:*:*	pkg:maven/org.springframework/spring-tx@5.3.12	CRITICAL	5

Рис. 7. Результат сканування SCA

OWASP Zed Attack Proxy (ZAP) являється інструментом для тестування на проникнення з відкритим вихідним кодом. ZAP розроблений спеціально для тестування веб-додатків і є одночасно гнучким та розширюваним. У нашому випадку, ми використовуємо контейнеризовану версію сканеру, яка запускає павука ZAP для зазначеної мети протягом (за замовчуванням) 1 хвилини, а потім чекає на завершення пасивного сканування, перш ніж повідомити про результати. Це означає, що скрипт не виконує жодних реальних «атак» і працюватиме протягом відносно короткого періоду часу (максимум кілька хвилин). На рисунку 8 зображено знайдені вразливості, їх кількість та рівень ризику.

Якщо проект готовий випускати свій продукт на ринок, необхідно запустити процес доставки програмного коду до хмарного середовища. Розробник відправляє зміни до репозиторію github, який повідомляє CD-процес Jenkins, про те, що необхідно створити контейнер релізної версії додатку, відправити його до сховища образів ECR, оновити задачу і відправити її до ECS. Дана задача, говорить сервісу контейнерів використовувати нову версію образу. На рисунку 9 зображено процес розгортання коду в AWS.

ZAP Scanning Report

Site: http://petclinic-lb-demo-1860407745.us-east-1.elb.amazonaws.com:8080

Generated on Wed, 4 May 2022 14:41:32

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	3
Low	5
Informational	2
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
Absence of Anti-CSRF Tokens	Medium	4
Content Security Policy (CSP) Header Not Set	Medium	9
Missing Anti-clickjacking Header	Medium	9
Application Error Disclosure	Low	1
Cross-Domain JavaScript Source File Inclusion	Low	10
Information Disclosure - Debug Error Messages	Low	1
Timestamp Disclosure - Unix	Low	2
X-Content-Type-Options Header Missing	Low	11
Information Disclosure - Suspicious Comments	Informational	1
User Controllable HTML Element Attribute (Potential XSS)	Informational	6

Рис. 8. Результат виконання скану OWASP ZAP

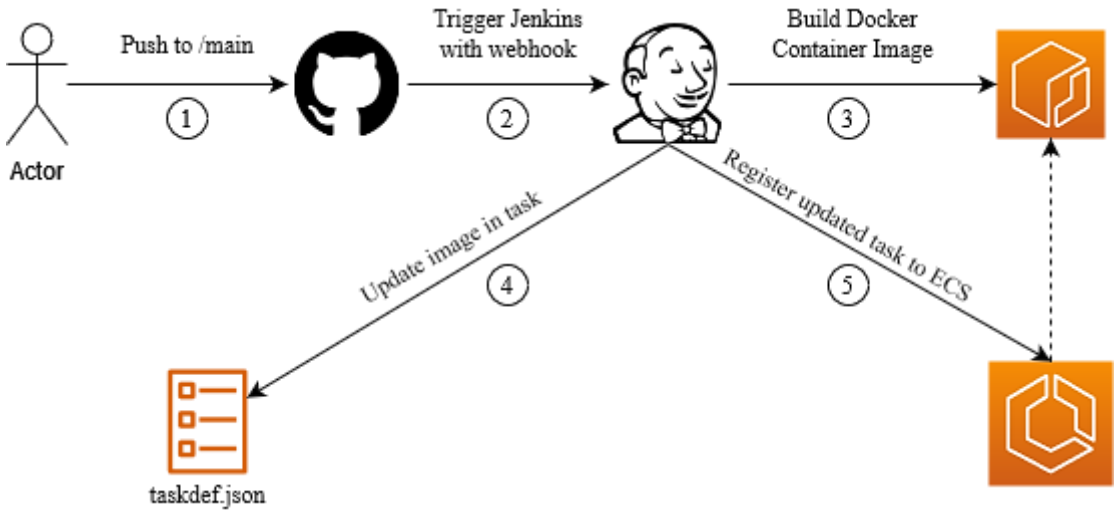

Рис. 9. Загальна схема випуску веб-ресурсу у зовнішнє середовище

Як можна побачити, сканування приносять багато корисної інформації для розробників, щодо використання вразливих залежностей, а також виявлених в результаті розробки та конфігурації вразливостей. В залежності від степені ризику у команди є часові ліміти для вирішення виявлених проблем. Якщо знайдена вразливість не високого рівня, а вирішення даної проблеми потребує багато часу, то бізнес може прийняти ризик або відтермінувати вирішення проблеми.

Розробка рекомендацій, щодо підвищення захищеності веб-ресурсу.

Хакери знаходять нові лазівки у кожному релізі програми, тому відсутність оновлень з останніми виправленнями, може призвести до появи вразливостей і нових векторів атак для зловмисників. Для більш якісної безпеки можна закрити версії різних веб сервісів від сторонніх очей. Це навряд зупинить зловмисника, але йому буде необхідно прикласти більше

зусиль та часу для вивчення вашого ресурсу. Іншою загальною рекомендацією, якою мають користуватися всі – збір і зберігання журналів. На етапі аналізу, можна налаштувати конвеєр на виявлення аномалій, вторгнення та шаблонів атак і відправити повідомлення експерту з безпеки або виконати певні дії для знешкодження проблеми автоматично. Після аналізу даних, всі журнали зберігаються в базі даних. Компоненти веб-ресурсів генерують велику кількість логів, тому треба відповідально відноситися як до критеріїв збору логів, так і до місця в якому вони зберігаються. Дані журналів додатків відіграють основну роль у стратегії виявлення вторгнень та аномалій. Системне журналювання часто не охоплює того, що розробники вважають вартим журналювання, а адміністратори не можуть домогтися, щоб журнали задовольняли всі їхні потреби. Тому необхідно впевнитися в існуванні єдиного структурованого формату журналів (JSON, XML, CSV), форматі часових міток, вказані джерела подій (ідентифікатор PID, публічний IP, додаток, вузел). Організація OWASP створила список рекомендованих для журналювання подій:

- Невдачі при валідації введення, наприклад, порушення протоколів, неприйнятні кодування, не валідні імена та значення параметрів.
- Невдачі при валідації виводу, такі як невідповідність набору записів бази даних, не валідне кодування даних.
- Вдалі та невдалі спроби автентифікації.
- Невдачі при авторизації (управління правами доступу).
- Невдачі при обслуговуванні сесій, наприклад модифікації ідентифікаційних значень у сесії з cookie.
- Помилки програми та системні події, такі як помилки синтаксису та часи виконання, помилки з'єднань, проблеми продуктивності, повідомлення про помилки сторонніх сервісів, помилки файлової системи, виявлення вірусу під час завантаження файлу, зміни конфігурації.

Журнали часто містять конфіденційну інформацію про організацію та її клієнтів. Найчастіше в них зустрічаються внутрішні вхідні дані чи паролі кінцевих користувачів, які знаходяться у повідомленні і не зашифровані належним чином. Часто трапляються випадки знаходження ключів API, які передаються у рядку HTTP GET-запиту та логування веб-серверами в їх журналах доступу. Це безперечно не публічні дані і вам варто зберігати необроблені журнали в безпечному місці. Іншою причиною необхідності захищати доступ до журналів є те, що зловмисники будуть шукати способи видалення слідів своєї активності при вторгненні в інфраструктуру. Ніхто, крім деяких адміністраторів, не повинен мати можливості видаляти дані журналів.

Інша рекомендація стосується надання найменших привілеїв для виконання необхідних функцій. Необхідно налаштувати дозволи і привілеї. Дозволи мережевих служб та дозволи файлів відіграють вирішальну роль у вашій мережевій безпеці. Якщо веб-сервер скомпрометований за допомогою програмного забезпечення мережевої служби, зловмисник може використовувати будь-який обліковий запис, запущений мережевою службою, для виконання завдань. Через це, просте встановлення мінімальних привілеїв користувачів для доступу до файлів веб-ресурсів та серверних баз даних, може сприяти запобіганню втрати або маніпулюванню даними.

Наступною рекомендацією є використання системи контролю версій, такої як Git. Як зазначалося раніше, розробники постійно вносять зміни до коду, тому необхідно впевнитися в тому, що ці зміни декларуються і вразі виявлення вразливостей відійти на попередню версію коду. Крім того, наявність централізованого сховища стає важливою частиною створення CI процесів. Github також надає можливість безпечно зберігати свої секрети у вашому репозиторії, тим самим підвищити захищеність коду. Дані секрети можуть бути використані під час розгортання CI процесів за допомогою Github Actions. Необхідно впевнитися, що користувачі використовують мультифакторну автентифікацію. Блокуйте конфіденційні дані, що передаються в GitHub за допомогою git-secrets або подібних до них,

як git pre-commit hook. Деякі файли в проекті не мають відслідковуватися в репозиторії. Необхідно визначити перелік цих файлів і внести список у .gitignore файл.

Також дуже важливим є впровадження захищеного циклу розробки програмного забезпечення. Необхідно строго слідувати за кожним етапом і відповідально відносити до проектування, реалізації, перевірки і підтримки додатку. Важливо надати розуміння команді, навіщо впроваджувати дану методологію, провести тренінги. На етапі розробки критично важливо впровадити цикл неперервної інтеграції для автоматизації тестування, пошуку вразливостей, розгортання додатку. Потрібно комбінувати інструменти пошуку вразливостей і періодично їх запускати. Іншою рекомендацією є захист каналу зв'язку між користувачем та веб-сервером. Необхідно впевнитися, що комунікація залишається конфіденційною, а дані цілісними. З цим може допомогти метод шифрування TLS, який впевнюється в тому, що пакети не можуть бути розшифрованими третьою стороною, а також, що будь-які зміни будуть помічені.

Дуже важливим є використання брандмауера, який призначений для виявлення та блокування сучасних атак на веб-ресурси. Його використання, може захистити веб-ресурс від наступних атак: SQL ін'єкції, віддалене виконання коду, міжсайтовий скриптинг, міжстайтове підроблення запитів, віддалене включення RFI, локальне включення LFI, обхід авторизації, небезпечні прямі посилання на об'єкти, перебір паролів. Треба зазначити, що проблеми безпеки веб-ресурсу WAF не вирішує, він лише зупинить частину запитів зловмисника до програми, він визначає лише відомі йому вразливості, оскільки має заздалегідь налаштовані сигнатури. Він не зможе захистити від варіацій вразливостей та експлойтів.

Висновки

У даній статті проаналізовано такі засоби виявлення вразливостей, як SAST, DAST, SCA, їх впровадження в безпечний SDLC, переваги та недоліки. Детально розглянуто питання влаштування сканерів вразливостей Dependency-Check та OWASP ZAP в процеси неперервної інтеграції (CI) та неперервної доставки (CD) для автоматизації виявлення вразливостей. Розглянуто процес створення інфраструктури в хмарному середовищі AWS з використанням інструменту IaC Terraform та налаштування CI/CD процесів за допомогою Jenkins. Розроблено рекомендації, щодо підвищення захищеності веб-ресурсу, серед яких: оновлення програмного забезпечення, налаштування журналювання, резервування, надання користувачам і компонентам найменших привілеїв, використання системи контролю версій, CI інструментів, сканерів вразливостей та брандмауерів.

Перелік посилань

1. Malcolm McDonald Web security for developers / Malcolm McDonald - S. Fr. : No Starch Press, Inc., 2020.
2. Що таке CI/CD [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/what-is-ci-cd/
3. IaaS (Інфраструктура як сервіс) – Режим доступу до ресурсу: https://www.ibm.com/cloud/learn/iaas?msclkid=c5cded98cf9011ec942a9fd7b2dc4444
4. Вступ до пом'якшення ін'єкцій SQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwaresecured.com/introduction-to-sql-injection-mitigation/ https://owasp.org/www-project-proactive-controls/v3/en/c5-validate-inputs
5. Як запобігти атакам обходу каталогу [Електронний ресурс] – Режим доступу до ресурсу: https://www.invicti.com/blog/web-security/directory-path-traversal-attacks/
6. Атака «Людина посередині» (MITM) [Електронний ресурс] – Режим доступу до ресурсу: https://www.rapid7.com/fundamentals/man-in-the-middle-attacks/
7. Огляд ринку DAST 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://www.anti-malware.ru/analytics/Market_Analysis/DAST-Market-Review-2021
8. OWASP Dependency-Check [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/www-project-dependency-check/

Надійшла: 06.06.2022 Рецензент: д.т.н., професор Гайдур Г.І.